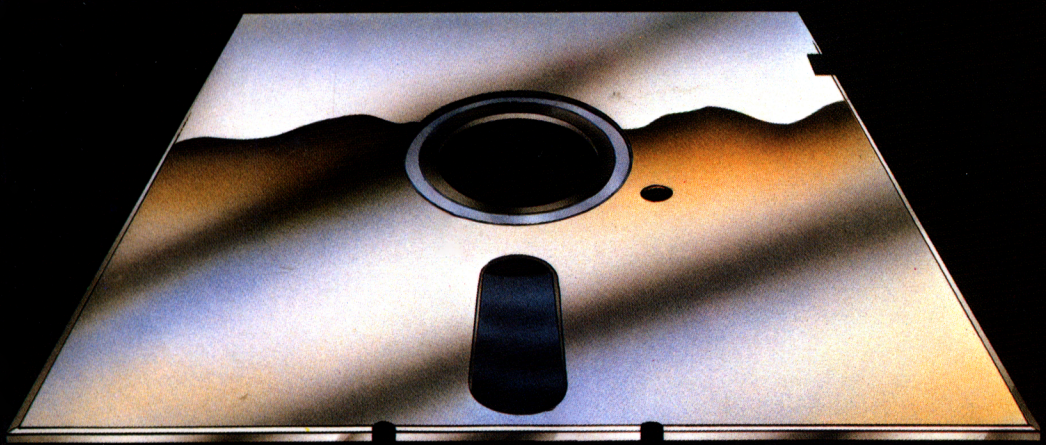
DISK BASIC

ON YOUR MICRO



Michael Chadwick & John Adrian Hannah

Sigma Press

DISK BASIC
On Your Micro

Michael Chadwick

and

John Adrian Hannah

Σ
SIGMA
PRESS

ISBN 0 905104 83 8

Published by:

SIGMA PRESS,
5 Alton Road,
Wilmslow,
Cheshire,
U.K.

Distributors:

U.K., Europe, Africa:
JOHN WILEY & SONS LIMITED
Baffins Lane, Chichester,
West Sussex, England.

Australia:
JOHN WILEY & SONS INC.,
GPO Box 859, Brisbane,
Queensland 40001, Australia.

Illustrations

These were prepared by Berit Sturn.

Acknowledgments

The following are all trade marks or registered trading names:

Commodore	MS-DOS
VIC-20	PC-DOS
Commodore 64	IBM-PC
Apple	BBC Computer
Applesoft	BBC BASIC
Apple II	

Any reference to any such name does not imply endorsement.

Printed in Great Britain
by J. W. Arrowsmith Ltd., Bristol

Introduction

In recent years the development of inexpensive home and personal computers, which can be programmed in BASIC, has experienced a tremendous increase. It will not be long before the computer has gained as firm a place in the household as the television set and the telephone.

Computer users can be divided into two groups. Some buy ready-made program packages which perform specified tasks. They learn to use the computer with these programs in the same way as the use of a washing-machine or a stereo is learned. For them, the computer is a machine for solving a particular problem.

Others go quite another way about it. They regard the computer as a partner with many talents and the will to learn, and make it their goal to teach it as much as possible - in other words, to program it. We are proceeding on the assumption that you, the reader, belong to this second group.

You have set yourself a high target and can work at it all your life, as the process of invention of newer and better programs in infinite. We will assume that you have already managed the first steps in the art of programming successfully, and have a command of the fundamental knowledge of BASIC. We further assume that you either have your own BASIC computer, or have access to one.

There are three possible reasons for your having decided to read this book:

1. You wish to learn some principles of the use of diskettes in connection with BASIC computers,
2. You are planning to purchase a diskette drive in due course (or better still, to have someone buy you one!),
3. You already possess a diskette drive, and would like to know how to use it.

In any case, this book will give you the advice you need. On the one hand you will learn the general principles of file handling with a diskette; on the other, how to write useful programs with particular reference to the four most popular BASIC dialects:

1. Commodore BASIC with a VIC-20 or Commodore 64 and a 1541 Disk Drive.
2. Applesoft on the Apple II+, Apple IIe or Apple IIc.
3. Microsoft BASIC: MAS-DOS or PC-DOS (on an IBM-PC).
4. BBC BASIC.

If your computer understands one of these versions of BASIC you can use the programs in this book without any alterations. If your computer uses a different dialect, a few small modifications will be necessary. These you can learn very quickly.

Contents

1.	**The Many Things You Can Do with a Diskette**		**1**
	1.1	The First Impression	1
		Treatment of Diskettes	1
	1.2	The Organisation of a Diskette	2
	1.3	What Advantages does the Diskette Have?	5
	1.4	Using the Diskette for Storing Programs	7
	1.5	What is a File?	9
	1.6	The Differences Between Various BASIC Systems	12
2.	**Simple Commands for Sequential Files**		**13**
	2.1	The OPEN Command	13
	2.2	The CLOSE Command	16
	2.3	How to Write Data to a File	17
	2.4	How to Read Data from a File	19
	2.5	Example: Your Friends' Phone Numbers	20
3.	**Unsorted Sequential Files**		**23**
	3.1	Example: A Phone Directory	23
	3.2	The Program Menu	26
	3.3	Creating the file PFILE	29
	3.4	Reading and Printing from the file PFILE	30
	3.5	Adding new Records	32
	3.6	Finding a Record	35
	3.7	Deleting a Record	37
	3.8	Copying the file PFILE	41
	3.9	"BBC Subroutine 10000"	42
4.	**Sorted Sequential Files**		**43**
	4.1	Example: A School Class List	43
	4.2	Creating the file CFILE	45
	4.3	Reading from and printing the file CFILE	46
	4.4	Inserting a New Record	47
	4.5	Finding a Record	50
	4.6	Entering the Grades	51
	4.7	Drawing up Statistics	54
	4.8	Deleting Grades	61

5. Random Access Files 63

5.1 Fundamentals 63
5.2 Example: A Stock Inventory 71
5.3 Initializing the file IFILE 75
5.4 Create New Entry 76
5.5 Printing the Inventory File 78
5.6 Gaining Access to an Article 80

6. Index Sequential Files 83

6.1 Example: A Book Catalogue 83
6.2 Initializing the file BFILE 89
6.3 Entering a New Book 90
6.4 Printing Out the Unsorted Book Catalogue 95
6.5 Printing Out the Sorted Book Catalogue 95
6.6 Finding Books by Author 97

7. Linked Lists 99

7.1 Principles 99
7.2 Example: A Customer Index 101
7.3 Initializing the File CLFILE 107
7.4 Inserting the Name of a New Customer 108
7.5 Deleting a Customer's Name 116
7.6 Printing Out the Complete List 118
7.7 Printing Out the Group List 119
7.8 Finding a Customer's Name 119

8. Files With Scattered Storage 121

8.1 Example: An Inventory File 121
8.2 Initializing the Files SFILE and OFILE 125
8.3 Entering a New Part 126
8.4 Unsorted Printing of the Files SFILE and OFILE 130
8.5 Access One Part 131

Appendix: File Handling Commands 134

Index 140

Chapter 1

The Many Things You Can Do with a Diskette

1.1 The First Impression

Have you a disk at hand? If not, have a look at Figure 1.1. This shows the external arrangement of a diskette; a circular plate covered in a layer of magnetizable substance, protected by a sleeve. Through the oval hole in the sleeve, the head can contact the diskette to read from it or write on it. The large round hole in the middle is the **drive aperture**, through which the disk is turned at about 300 r.p.m. The position of the disk can be determined by means of the small round hole in the disk; when this coincides with the small round hole in the sleeve, once every revolution, a light signal passes through.

At the top of the right-hand edge is the **write** protect notch, which can be masked to prevent accidental deletion, as on a tape cassette. Make sure from your manual whether in the case of your computer this protection is achieved by masking the indentation or by leaving it open.

Taking a diskette in your hand, you will see that it is much more flexible than an ordinary record. This is why it is also known as a **Floppy Disk**. Handle it carefully, and do not try to bend it.

There are various standard sizes of disk. The most usual kind for the home computer is the 5¼ inch diskette, also called the mini-diskette. We will be concerned exclusively with this size.

Treatment of Diskettes

If you treat your diskette properly, it should give you many years of good service. Please observe the following rules:

1. When it is not in use, always keep the diskette in its paper sleeve, which protects it from dust.

2. Do not expose the diskette to conditions such as extreme temperatures, direct sunlight, moisture etc.

3. Keep the diskette away from magnetic fields, which could delete the stored data. Do not lay it on the drive housing or on your computer.

4. Never touch the magnetic layer with your fingers; do not attempt to clean the surface.

5. Use only a soft felt-tip to write on the sleeve label, not a pencil or a ball-point.

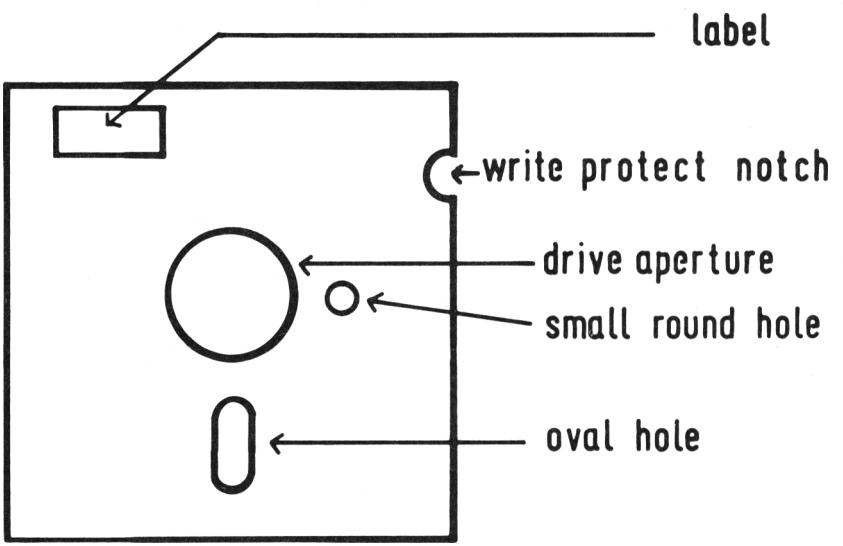


Fig. 1.1; external arrangement of a diskette

1.2 The Organization of a Diskette

You are now familiar with the external appearance of a diskette. What you cannot see is its internal structure or organization. When you buy a new diskette at the shop it is completely empty, like a sheet of paper without lines or page numbers. The computer can do nothing with this diskette in its present form. First of all an organizational framework must be composed. This process is called **formatting** or **initialization**. Here the diskette is divided into **tracks and sectors**. A special **formatting program**,

provided by the computer manufacturer, does this job (consult your manual on this point). We refer to this as **software sectoring** or **soft-sectoring**, as the division into sectors is provided by the software, i.e., by means of a program, as opposed to **hard-sectored diskettes**, which we will not be dealing with. Figure 1.2 shows an example of the division of a diskette into 16 sectors. The number of sectors depends on the computer being used, and ranges from 10 to 16.

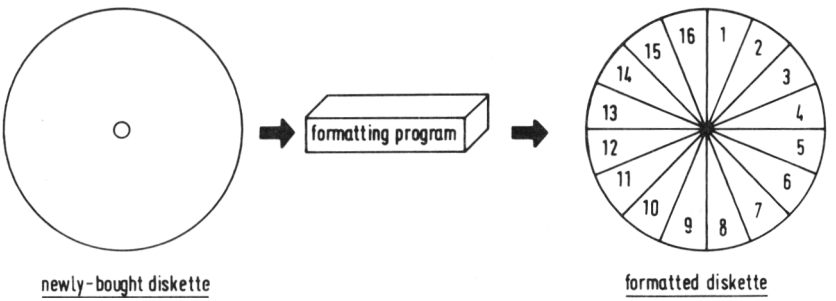


Fig. 1.2; division of a diskette into 16 sectors

You can imagine the sectoring as being very much like the cutting up of a big cake into individual slices. The formatting program marks the surface of the diskette into sector boundaries; individual areas are thus achieved, resembling, for example, the division of a city into postal areas. You will want to store information on your diskette and to find this information again later. It is therefore necessary to give each amount of information a kind of **address**. A part of this address is the **sector number**.

The second part is the **track number**. A diskette is divided into concentric circles called **tracks**, the number of which depends on the manufacturer; usually there are between 30 and 40 tracks. Figure 1.3 shows, by way of illustration, a diskette with 40 tracks.

Each piece of information can be clearly addressed by giving it a sector and a track number. Figure 1.4 shows a section of a diskette; the information called for can be found in Sector 7, Track 2.

Using a diskette, therefore, enables you to locate any of the stored information easily. Equally, a diskette is a memory with **direct access**, while magnetic tapes and tape cassettes do not allow direct access. We will discuss this point in more detail later.

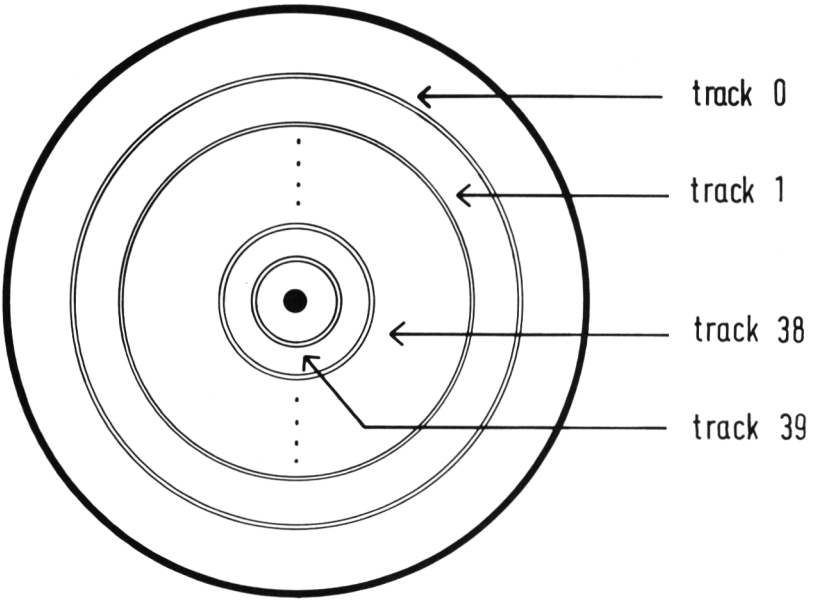


Fig. 1.3; division of a diskette into 40 tracks

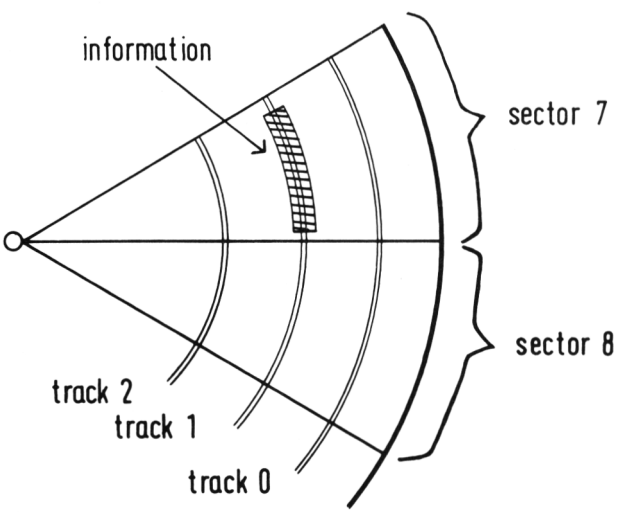


Fig. 1.4; section of a diskette

1.3 What advantages does the diskette have?

The diskette is an external memory medium which stores data or programs permanently by magnetic recording. With the help of diskettes, you can store as much information as you like **for as long as you like**. Most home computers delete all data and programs in the workspace when they are switched off, and everything has to be entered into the computer again the next time it is switched on. The diskette saves you this senseless work. The primary and most important advantage therefore lies in the **permanent storage** of data.

The second advantage lies in the **storage capacity**. This is given in **bytes** or **kilobytes**; one character can be stored in a byte. One kilobyte = 1KB = 1024 bytes.

The capacity of the workspace in most home computers varies a lot - it can be as little as one KB or as much as 64 KB. The amount frequently met with is 32 KB. In a workspace of this capacity you can store up to 32,000 characters - about 10 to 20 typewritten pages. For many data-processing problems, this is too small. An additional **external memory** with greater capacity is needed. The diskette is an inexpensive solution which suffices in many cases.

How many characters can be stored on a diskette? This depends on the number of sectors and tracks, and on the number of bytes per sector and track.

For example: 256 bytes per sector and track
40 tracks
16 sectors

capacity 256 x 40 x 16 = 160,000 characters = 160 KB

With 160 KB, the diskette provides five times the capacity of a workspace with 32 KB. Using double density and both sides of the diskette allows the capacity to be increased considerably.

The number of diskettes which you can use for storage of your information depends solely on your wallet; in principle, the size of the memory bank at your disposal is entirely up to you. (See figure 1.5). In this manner, you can build up a comprehensive program library over a period of time; you can collect diskettes as other people collect records or stamps. .

What can be stored on diskettes? There are three main groups:

1. commercial software, 2. your own programs, 3. data

Commercial software includes all those programs which you can buy, e.g. operating systems, translators, text processing programs, games, and so forth. As the owner of a diskette drive, the wide world of software is open to you. Perhaps you would like to work with another programming language, such as PASCAL? You merely buy a diskette containing the PASCAL translator, load this into your workspace, and your computer

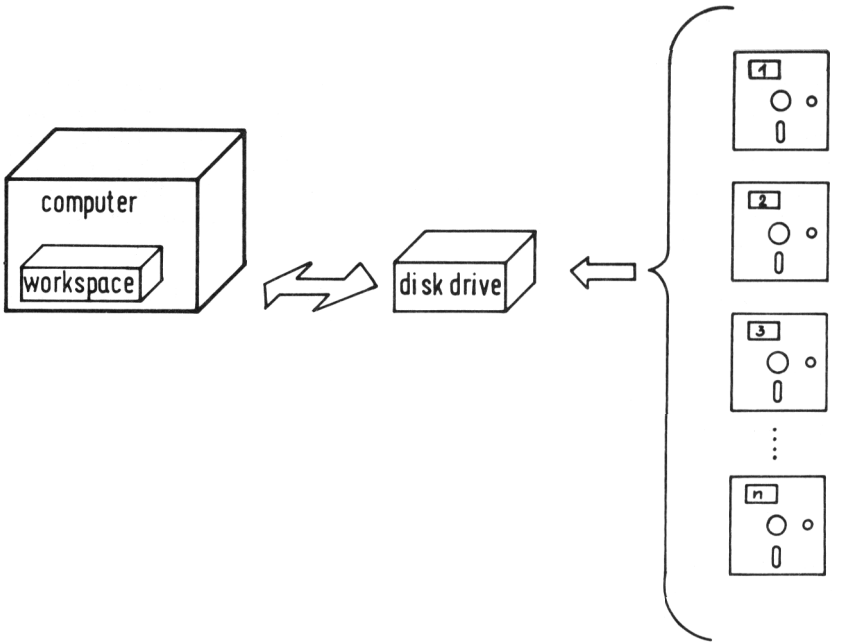


Fig.1.5; the external memory bank is as large as you like

understands PASCAL. Or perhaps you need a calculating program, such as VisiCalc, SuperCalc or Multiplan; no problem with a diskette drive! These examples alone show that you can only work with commercial software if you have at least one diskette drive at your disposal.

Again, there is the possibility of using your own programs. Everyone who invents his own programs, and would like to take this activity seriously, should be able to store his programs permanently. In principle, this is of course possible with a cassette recorder, but the amount of time wasted in waiting soon gets on your nerves; sooner or later you decide on a diskette.

Finally, a diskette serves for the storage of data. Let us suppose you wish to keep a stock inventory using a computer. About 1,000 articles are to be stored, 100 characters per article. You therefore need a memory with around 100,000 bytes, or 100 KB. The diskette has enough room for this amount, and also permits direct access to the data, so that, for example, the stock of article No. 177 can be read directly from the diskette, which is not possible using magnetic tapes or cassettes.

To sum up, here are the advantages of the diskette in a few words:

1. Permanent storage
2. Large capacity
3. Relatively low price
4. 'Gateway to the world of software'
5. Possibility of building up your own program library
6. Storage of large amounts of data

The diskette has become the most popular external memory for use at home or in smaller or medium-sized businesses. Its limits become apparent when very large amounts of data are to be processed; for such purposes a hard disk is used.

1.4 Using the Diskette for Storing Programs

Perhaps you have bought a home computer without peripherals, and thus gained your first experience of programming with BASIC. You must certainly have been annoyed often enough to have your painstakingly-entered program deleted when the computer was switched off.

The use of a cassette recorder is of some help here. Your programs can be permanently stored on inexpensive cassettes. You can live with this system for some time - until the day your patience is exhausted, and loading and saving larger programs becomes tedious.

These problems need no longer worry you with a diskette. Even with larger programs, the time spent on waiting for transport to and from the workspace is minimal. There is also no need to think about where your program is to be found on the diskette. You can forget about 'fast forward' and 'rewind'. Figure 1.6 shows both transport directions and their terms. These are also the commands for transport:

SAVE: The program is transferred from the workspace to the diskette

LOAD: The program is transferred from the diskette to the workspace.

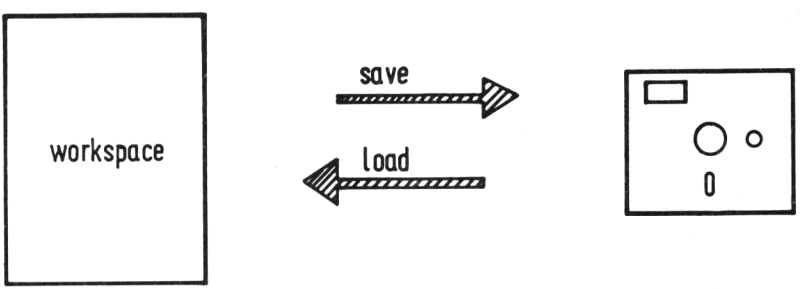


Fig. 1.6; load and save

Unfortunately, the syntax is not identical for all computers. The following examples show what commands have to be given to store a program which we shall call 'TEST' on the diskette:

Applesoft: SAVE TEST,D2
Microsoft: SAVE "TEST"
CBM: SAVE "TEST",8
BBC: SAVE "TEST"

You give the equivalent commands for LOAD. Should you have two drives, you can include in the commands which drive is to be used:

Applesoft: SAVE TEST,D2
Microsoft: SAVE "2:TEST" (or: SAVE"B:TEST")
CBM: SAVE"2:TEST",8
BBC: SAVE":1.TEST"

The above commands cause the program to be stored on a diskette in the second drive.

If you wish to see the file directory of the diskette, you must give the appropriate commands, which are unfortunately not identical for all computers:

Applesoft: CATALOG (or: CATALOG,D2)
Microsoft: FILES (or: FILES-2)
CBM: LOAD"$",8, then:LIST
BBC: *CAT (or*CAT 1)

Note that the command LOAD in the CBM version of BASIC will delete any program in the workspace. To prevent this, you should store it on a diskette beforehand.

Finally, it is necessary to know how to delete a program from the diskette. Use the following commands to delete the program 'TEST':

Applesoft: DELETE TEST (or: DELETE TEST,D2)
Microsoft: KILL"TEST" (or: KILL"2:TEST")
CBM: PRINT#1,"S:TEST" (S stands for "scratch")
BBC: *DELETE TEST

In practice, the procedure is to type a BASIC program and to store this original version on the diskette straight away. Then begin removing errors from the program - there are bound to be some; this is called debugging. You then have a second version, which you also store on the diskette. There are two possible ways of doing this:

1. Give the second version a different name to the first. For example, if the first is called TEST1, the second can be called TEST2. In this way, both versions of the program can be stored on the diskette.

2. If you no longer need the first version, store the second version **under the same name**. The first version will thus be overwritten.

(Note for CBM-BASIC: if you wish to overwrite the program TEST1 with the second version, the command to be given is: SAVE"@: TEST",8

Now make yourself familiar with the different commands. Put a formatted diskette into the drive. Type a short BASIC program and then store it on the diskette. If you have two drives, use two diskettes. Look at the file directory of the diskette. Delete the program in the workspace. Transfer the program back from the diskette to the workspace. Check by means of the commands LIST and RUN that your program is back again. Finally, delete it from the diskette.

1.5 What is a File?

We have already mentioned the term "file handling" without defining the word "file". Let us do so now.

A file is an amount of related data, which, in our case, is stored on a diskette.

It is structured hierarchically. Figure 1.7 shows that a file consists of individual **records**. These again are composed of **fields**. A field is therefore the smallest unit of a file.

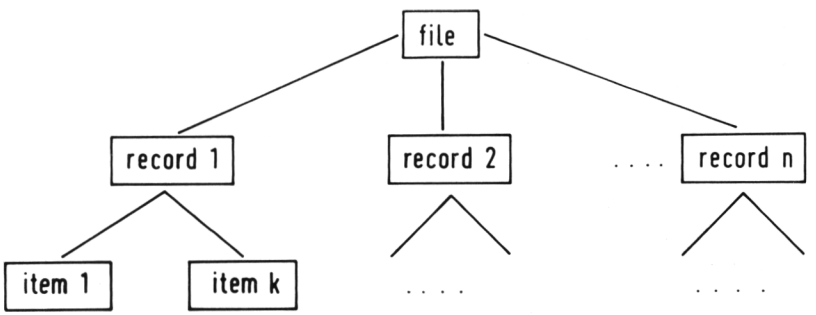


Fig.1.7; structure of a file

The following examples show how a record can be divided into several fields:

File	Fields
Inventory	number of article, specification, amount, price
Personnel file	personnel number, name, address, salary group
customer file	customer's number, name, address

Addresses can be divided into four further fields, namely, house number, street, town and postal code. These examples enable you to imagine what a dominant part such files play in practice. The usual card indexes can be replaced by an "electronic card index", namely the file (see Figure 1.8).

Files differ from one another in respect of their organization. This is determined by the **storage form** and the access mode. There are four of the former and two of the latter (see Figure 1.9).

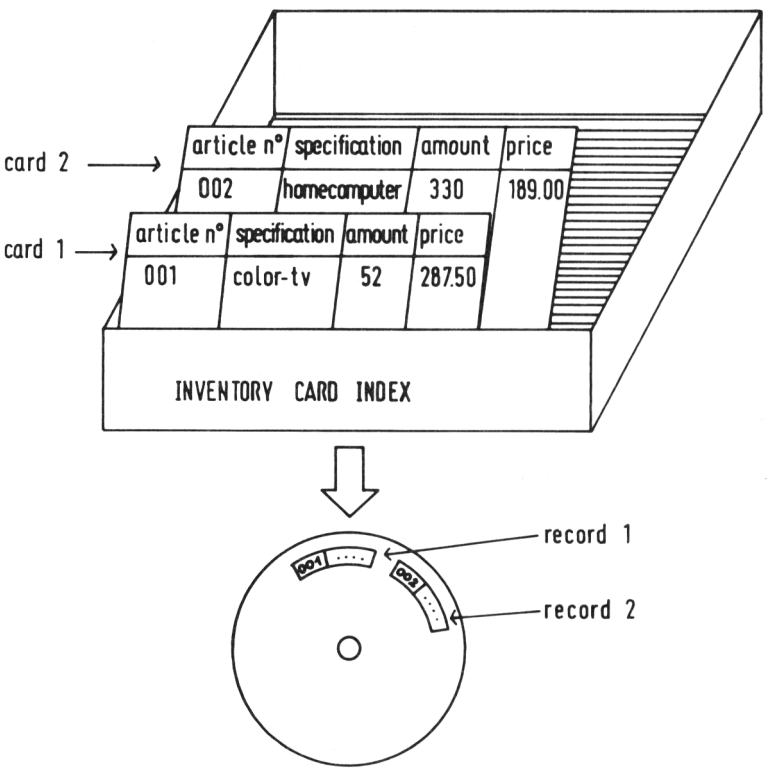


Fig.1.8; The "electronic card index"

Storage forms:	sequential
	scattered
	index sequential
	linked
Access modes:	indirect (sequential)
	direct (random)

You will get to know the storage forms and access modes in more detail in chapters 3 to 8. The storage form indicates the way in which the individual records are stored in the file. The access mode describes how to retrieve records.

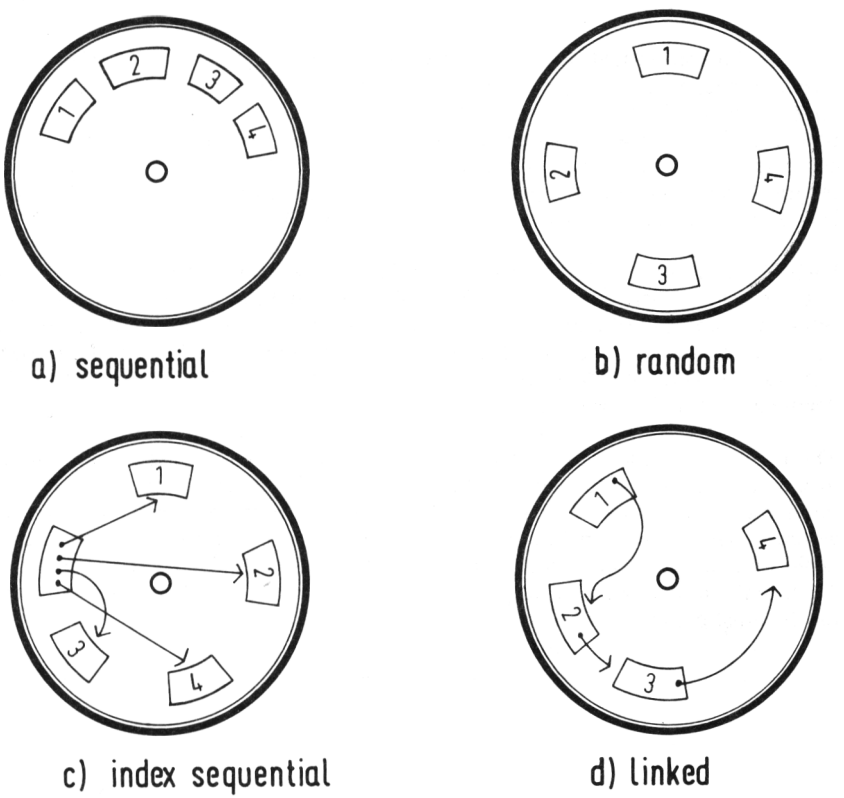


Fig.1.9; Four possible storage forms

Your choice of organization form depends on the purpose of the file and the amount of programming you feel you can take on. By means of typical examples you will get to know the various organization forms, and will then be in a position to choose the one best suited to your purposes, and be able to organize a program accordingly.

The differentiation is often made between **programs** and **data**. Programs are instructions to a computer to perform certain tasks; they are **control information**. Data, on the other hand, is either **classifying information** or **quantity information**. Examples of classifying information are names, account numbers, or street names; examples of quantity information are prices, numbers of pieces, or dimensions. When storing on a diskette, however, it is immaterial whether we are concerned with programs or with data, as both are really **character strings**, consisting of letters, digits or special symbols. Thus programs are stored on the diskette in the form of files. With the command SAVE, a program file is created. However, from now on we will only be dealing with "real" files, i.e., those containing data.

1.6 The differences between various BASIC systems

Like all problem-oriented programming languages, BASIC was conceived as a machine-independent language. Thus a BASIC program must theoretically be operable on any BASIC computer. The fact that this is, unfortunately, not the case in practice will make itself apparent to you as soon as you try to run a program on your computer which was not written specifically for your type of machine, such as a program taken from a book or periodical. As a rule, your computer will report "SYNTAX ERROR" when it receives any BASIC commands which it does not know.

BASIC commands can be divided into two groups: a **language nucleus** with basic instructions which are understood or interpreted equally by all BASIC computers, and **extensions and modifications** which differ from one manufacturer to the next. The language nucleus includes, for example, the commands LET, PRINT, FOR, IF-THEN, GOSUB, and RETURN. Most other commands - particularly those necessary for file handling - are unfortunately not uniform.

This fact has determined the form of this book. It does not make sense to limit yourself to one single BASIC system and to demonstrate only the file processing instructions of this one system. It is very probable that you will be confronted with a different BASIC system sooner or later. On the other hand, not all the systems currently available can be dealt with, as there are simply too many. We have considered it a reasonable compromise to start with the generally-valid principles and then to develop programs for the most popular BASIC systems.

Chapter 2

Simple commands for sequential files

2.1 The OPEN-command

Before you can begin work on a file, you must open it, and the OPEN-command does this. You give the following information:

1. File name
2. Processing mode: reading or writing
3. Allocation of a logical file number
4. Allocation of a channel number

The first two pieces of information must be given in all BASIC versions; the others are optional. The file name is a character string consisting of alphanumerical characters (i.e., letters and digits), and must begin with a letter. The maximum length varies, but we are going to restrict ourselves to seven characters.

There are two different processing modes: reading and writing. Figure 2.1 shows the data flow between workspace and diskette in both cases.

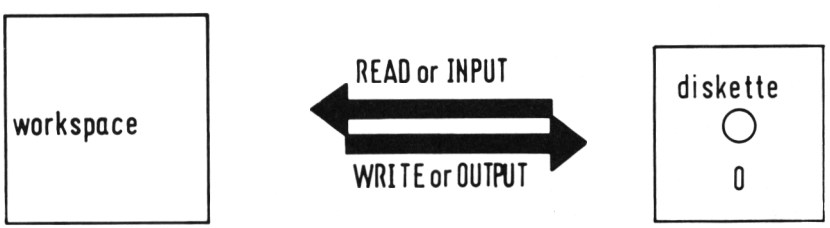


Fig. 2.1; reading and writing

If you want to create a new file, this has to be opened for writing. The new name is entered in the file directory. If there is already a file with this same name, it will be deleted.

If you want to read a file, then open it using the "read" mode. The file must of course already exist in this case. You can then read one record after another into the workspace and process it further.

In some BASIC systems, each file is allocated a logical file number, and the file is then **addressed** via this number. Sometimes a channel number must be given as well.

Let us now consider the individual OPEN-commands in more detail. The **Reading Mode** is designated by READ or INPUT, the **Writing Mode** by WRITE or OUTPUT.

Opening for reading

Applesoft: PRINT CHR$(4);"OPEN";filename
PRINT CHR$(4);"READ";filename
Microsoft: OPEN filename FOR INPUT AS # logical file number
CBM: OPEN logical file number,device number,channel number, filename, type, mode
BBC: channel number=OPENIN "filename"

Example:

The file TEST is to be opened for reading.

Applesoft: PRINT CHR$(4);"OPEN";"TEST"
PRINT CHR$(4);"READ";"TEST"
Microsoft: OPEN"TEST"FOR INPUT AS #3
CBM: OPEN 3,8,2,"TEST,S,R"
BBC: X=OPENIN"TEST"

Notes on the individual BASIC versions:

Applesoft:

As you see, there is no specific OPEN command. Opening is achieved by the PRINT command, which first of all shows the symbol CHR$(4). As this appears frequently, we will from now on use the abbreviation D$. So at the start of every program you must write the following command:

```
LET D$ = CHR$(4)
```

Using a second PRINT command, you can say whether the file is to be opened for reading or for writing. We can now write both commands in the shorter form:

```
PRINT D$;"OPEN";"TEST"
PRINT D$;"READ";"TEST"
```

The commands can now be shortened even further by assigning the file name to a string variable. This is possible with the other BASIC versions, too. For example, you can call the variable F$.

```
LET F$="TEST"
```

The PRINT commands are now:

```
PRINT D$;"OPEN";F$
PRINT D$;"READ";F$
```

As the file has not been allocated a logical file number, **all following INPUT commands refer to this file!** So, for example, if you give the command

```
INPUT X
```

your computer will not read from the keyboard, but from the file TEST. Please note this peculiarity of Applesoft-BASIC!

Microsoft

With the term FOR INPUT you indicate that the file is to be opened for reading. You give the file TEST the arbitrary logical number 3. From now on, access is gained with this number. The numbers usually vary from 1 to 15; we shall in future choose smaller values between 1 and 5.

CBM

Here we have also chosen 3 to be the logical file number. The device number of the Floppy Disk is 8. The channel number, also called the secondary address, we have designated as 2. The letter S indicates that the file is of the sequential type; R stands for READ. The secondary address can take values between 2 and 14.

BBC

The channel number variable is X; the reading mode is indicated by the supplementary IN.

Opening the file TEST for writing

Applesoft:
```
PRINT D$;"OPEN";"TEST"
PRINT D$;"WRITE";"TEST"
```
Microsoft:
```
OPEN"TEST"FOR OUTPUT AS#3
```
CBM:
```
OPEN 3,8,2,"TEST,S,W"
```
BBC:
```
X=OPENOUT"TEST"
```

Note for Microsoft-BASIC

Here we have a different mode for the OPEN command. It has the following form:

OPEN letter,file number, file name

You can choose one of the following letters:

O Open sequential file for output
I Open sequential file for input.
R Open random file for either input or output

For example, the following commands are identical:

```
OPEN "DATA" FOR OUTPUT AS#2
or
OPEN "O",#2,"DATA"
```

We shall only use the first version throughout this book.

2.2 The CLOSE Command

Unlike the OPENcommand, the CLOSE command is very simple. Except with Applesoft, you simply give the logical file number or channel number.

Applesoft: PRINT D$;"CLOSE";"TEST"
Microsoft: CLOSE 3
CBM: CLOSE 3
BBC: CLOSE#X

With these commands, the files which we opened using the last example are closed again. You must give the CLOSE command at the end of each file processing in order to complete it properly. If you forget to give it and later want to gain access to the file, the computer will indicate an error.

All file handling follows this general pattern:

Open file

read or write data records

close file

Compare this process with using a card index:

open box

read from or write on the card

close box

Let's now have a look at the basic commands for reading and writing.

2.3 How to write data to a file

We will assume you are already familiar with the PRINT command, which displays data on the screen. If you add a number symbol to the command word PRINT, you will have the writing command for sequential files:

PRINT# logical file number, list of variables.

Example:

On opening the file TEST, we have given it the logical file number 5. The variable X has the value 100. Using the command

```
PRINT#5,X
```

write the value 100 to the file TEST. If you then give a further PRINT# command, the new value will be entered to the right of the old, as an internal pointer is automatically moved one position further:

```
LET X=X+10
PRINT#5,X
```

The value 110 will be entered beside the value 100. The file now contains the following:

File TEST: 100↵ 110↵

The symbol " ↵ " represents the RETURN sign, which is produced after every PRINT command.

Note for Applesoft-BASIC

After a file has been opened for writing, every PRINT command causes output to this file. So the PRINT# command is not needed.
This method has two disadvantages:

1. You cannot at the same time output data on the screen and to the file;
2. Not more than one file can be opened for reading at one time; the same is true of writing.

The following program shows you how to write the first 20 natural numbers to a file which we shall call TWENTY.

Applesoft:

```
10 REM WRITE 20 NUMBERS
20 LET D$=CHR$(4)
30 PRINT D$;"OPEN";"TWENTY"
40 PRINT D$;"WRITE";"TWENTY"
50 FOR I=1 TO 20
60 PRINT I
```

```
70 NEXT I
80 PRINT D$;"CLOSE";"TWENTY"
90 END
```

Microsoft:

```
10 REM WRITE 20 NUMBERS
20 OPEN "TWENTY" FOR OUTPUT AS#1
30 FOR I=1 TO 20
40 PRINT#1,I
50 NEXT I
60 CLOSE 1
70 END
```

CBM

```
10 REM WRITE 20 NUMBERS
20 OPEN 1,8,2,"TWENTY,S,W"
30 FOR I=1 TO 20
40 PRINT#1,I
50 NEXT I
60 CLOSE 1
70 END
```

BBC

```
10 REM WRITE 20 NUMBERS
20 X=OPENOUT"TWENTY"
30 FOR I=1 TO 20
40 PRINT#X,I
50 NEXT I
60 CLOSE#X
70 END
```

If you compare the four programs, you will see that a considerable number of commands are identical. For this reason we will no longer give four complete programs, but one only, in which different versions are identified accordingly. You can then choose the version appropriate to your computer.

```
10 REM WRITE 20 NUMBERS
Applesoft:  20 LET D$=CHR$(4)
            21 PRINT D$;"OPEN";"TWENTY"
            22 PRINT D$;"WRITE";"TWENTY"
Microsoft:  20 OPEN "TWENTY" FOR INPUT AS#1
CBM:        20 OPEN 1,8,2,"TWENTY,S,W"
BBC:        20 X=OPENOUT"TWENTY"
30 FOR I=1 TO 20
Applesoft:  40 PRINT I
Microsoft, CBM: 40 PRINT#1,I
BBC:        40 PRINT#X,I
50 NEXT I
```

Applesoft:	60 PRINT D$;"CLOSE";"TWENTY"
Microsoft,CBM:	60 CLOSE 1
BBC:	60 CLOSE#X
	70 END

Now it's time to switch on your computer and enter your first program. Put in a new diskette and format it if necessary - your manual will show you how.

It is certainly useful to save the programs mentioned in this book on a diskette. Use either another diskette, or save both the programs and your files on the same one. In this case you will have to give the program a different name to that of the file. You can't, for example, use TWENTY, as a file with this name already exists. The simplest way is to put P for program in front of the name. The SAVE command then looks like this:

Applesoft: SAVE PTWENTY

Microsoft: SAVE"PTWENTY"

CBM: SAVE"PTWENTY",8

BBC: SAVE PTWENTY

2.4 How to read data from a file

If you want to check whether the numbers 1 to 20 have been properly saved in the file TWENTY, you must read the contents of this file. In analogy to the PRINT# command, the following command is used to read data from a file:

INPUT# logical file number, list of variables

Example:

The file NUMBERS has ben allocated the logical file number 4. It contains the numbers 10,20 and 30. Using the command:

INPUT#4,X

the number 10 is read from the file and assigned to the variable X. With the further command

INPUT#4,Y

the variable Y is given the value 20. You can, of course, read all three numbers with one single INPUT command:

INPUT#4,X,Y,Z

The variables X,Y,Z are given the values 10,20 and 30 respectively.

Note for Applesoft BASIC

The analogy described with the PRINT command is valid here; i.e., all INPUT commands refer to the file opened for reading.

The following program reads the numbers saved in the file TWENTY, and shows them on the screen (as the commands are now getting longer, we will abbreviate "Applesoft" to "Apple" and "Microsoft" to "MS"):

```
                10 REM READ 20 NUMBERS
Apple:          20 LET D$=CHR$(4)
                21 PRINT D$;"OPEN";"TWENTY"
                22 PRINT D$;"READ";"TWENTY"
MS:             20 OPEN "TWENTY" FOR INPUT AS#1
CBM:            20 OPEN1,8,2,"TWENTY,S,R"
BBC:            20 X=OPENIN"TWENTY"
                30 PRINT"CONTENTS OF FILE TWENTY"
                40 FOR I= 1 TO 20
Apple:          50 INPUT N
MS,CBM:         50 INPUT#1,N
BBC:            50 INPUT#X,N
                60 PRINT N
                70 NEXT I
Apple:          80 PRINT D$;"CLOSE";"TWENTY"
MS,CBM:         80 CLOSE 1
BBC:            80 CLOSE#X
                90 END
```

2.5 Example: Your Friends' Phone Numbers

Naturally, names as well as numbers can be saved in a file. Our first applied example shows you how to store a list in a file containing the first names and phone numbers of several friends. We will limit ourselves to ten names, and also assume that the numbers have a maximum of four digits. In the next chapter we will drop these limitations and develop a program with which you can store as many names as you like with phone numbers of any length.

The structure of the program is very simple. First, the file FRIENDS is opened for writing. Ten names and phone numbers are entered in a FOR-loop and written to the file. Please note that, in the Applesoft version, the command PRINT D$ must follow the output to the file, so that the next prompt, YOUR FRIEND'S NAME, is output on the screen and not to the file (see line 72)

After the names and phone numbers have been entered, the file **FRIENDS** is closed and then reopened for reading (lines 90 to 100). Then the list appears on the screen.

Please note that name and number are each written to the file by an individual PRINT command (line 70, line 71). This causes the RETURN symbol (↵) to be saved beside each name and each number:

File FRIENDS: DICK *2* ↵ 1123 ↵ BOB ↵ 332 ↵ etc.

The file thus consists of data records of varying length, each terminated with a RETURN.

```
                    10 REM YOUR FRIENDS' PHONE NUMBERS
Apple:              20 LET D$=CHR$(4)
                    21 PRINT D$;"OPEN";"FRIENDS"
MS:                 20 OPEN"FRIENDS"FOR OUTPUT AS#1
CBM:                20 OPEN 1,8,2,"FRIENDS,S,W"
BBC:                20 X=OPENOUT"FRIENDS"
                    30 PRINT "ENTER 10 NAMES AND PHONE
                       NUMBERS"
                    40 FOR I=1 TO 10
                    50 INPUT "YOUR FRIEND'S NAME";N$
                    60 INPUT "PHONE NUMBER";P
Apple:              70 PRINT D$;"WRITE";"FRIENDS"
                    71 PRINT N$:PRINT P
                    72 PRINT D$
MS,CBM:             70 PRINT#1,N$:PRINT#1,P
BBC:                70 PRINT#X,N$:PRINT#X,P
                    80 NEXT I
Apple:              90 PRINT D$;"CLOSE";"FRIENDS"
MS,CBM:             90 CLOSE 1
BBC:                90 CLOSE#X
Apple:              100 PRINT D$;"OPEN";"FRIENDS"
                    101 PRINT D$;"READ";"FRIENDS"
MS:                 100 OPEN"FRIENDS"FOR INPUT AS#1
CBM:                100 OPEN 1,8,2,"FRIENDS,S,R"
BBC:                100 X=OPENIN"FRIENDS"
                    110 PRINT"LIST OF PHONE NUMBERS"
                    120 PRINT "------------------"
                    130 FOR I=1 TO 10
Apple:              140 INPUT N$,P
MS,CBM:             140 INPUT#1,N$,P
BBC:                140 INPUT#X,N$,P
                    150 PRINT N$,P
                    160 NEXT I
Apple:              170 PRINT D$;"CLOSE";"FRIENDS"
MS,CBM:             170 CLOSE 1
BBC:                170 CLOSE#X
                    180 END
```

Figure 2.2 shows an example of a phone list. You might like to save this little program example on your diskette. Use the name PFRIENDS (program FRIENDS).

Apple:	SAVE PFRIENDS
MS:	SAVE"PFRIENDS"
CBM:	SAVE"PFRIENDS",8
BBC:	SAVE PFRIEND

LIST OF PHONE NUMBERS

DICK	1123
BOB	332
BILL	4455
LIZ	989
TOM	1156
JACK	3351
MARGARET	4445
MARY	5525
JIM	667
RON	333

Fig. 2.2; Example of a Phone List

Chapter 3

Unsorted Sequential Files

3.1 Example: A Phone Directory

In the last chapter, you met with a small program for setting up a phone directory. Now we're going to develop a program which works with a **menu** and performs the following tasks:

1. Creating the file
2. Reading and printing the file
3. Adding new records to the end of the file
4. Writing a record
5. Deleting a record
6. Copying the file

Before we begin the actual programming, we design the program and data structures. Figure 3.1 shows how the program is constructed. We will give it the name PHONE. The numbers in each box stand for the line numbers in the appropriate subroutine.

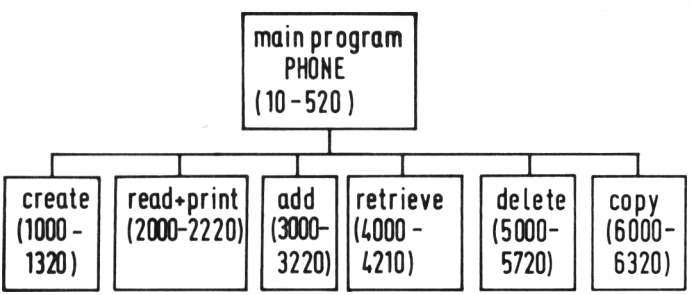


Fig. 3.1; Structure Chart for the Program PHONE

The data structures are very simple. Each record consists of two fields:

Name	Phone number

All records go to make up a file, which we shall call PFILE. Now there are several possible ways of organizing the file. Let's look at three of these more closely.

1st Method: Number of records is saved at the beginning of the file

Look at Figure 3.2. This shows a sample file of three records. The number of records is saved at the beginning, and this has the advantage of letting you know exactly how many records are contained in the file. You read the number and then construct a FOR-loop. The commands could, for example, be as follows:

```
INPUT#1,N
FOR I=1 TO N
INPUT#1,N$,P$ (read name and phone number)
NEXT I
```

The disadvantage of this method is that you have to update the number at the beginning of the file every time you add or delete a record.

(You need to use INPUT#X,N on the BBC, where X is the OPENIN channel).

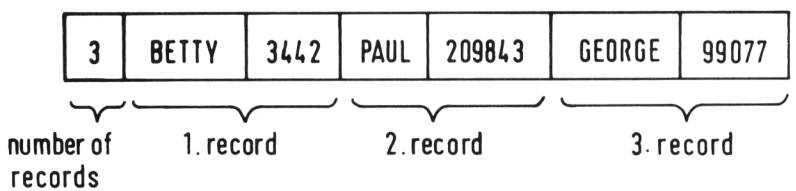


Fig.3.2;The number of records is shown at the beginning of the File

2nd Method: Special Marking at the End of the File

This method avoids the disadvantage just mentioned. Choose for the end marker a special record such as this:

*	0

Figure 3.3 shows what the file looks like in this case. You can read this file with the following commands:

```
100 INPUT#1,N$,P$
110 IF N$="*"THEN CLOSE...
      .
      .
      .     process data
      .
      .
200 GOTO 100
```

Unfortunately, this method, too, has a serious drawback. If you want to append new records to this file (i.e., add them to the end), the last record is in the way. It must be removed and appended to the "new" file.

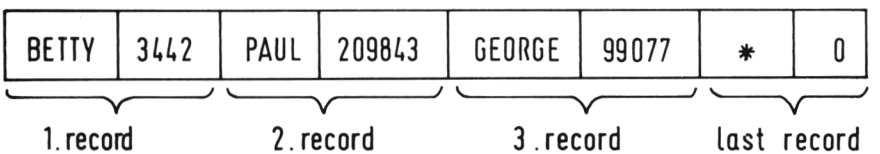


Fig.3.3; Special Marking at the End of the File

3rd Method: The Command EOF

Many BASIC systems contain the command EOF (end of file). The disk operating system (DOS) marks the end of a file with an EOF label (see Figure 3.4). When reading, you can ask by means of the EOF command whether the end of the file has been reached. These commands show the principle:

```
        100 IF EOF(1) THEN CLOSE...
BBC:    100 IF EOF(X) THEN CLOSE#X
        110 INPUT#1,N$,P$
              .
              .
              .  process data
              .
              .
        200 GOTO 100
```

This method avoids the disadvantages of the first two. There is no need to keep count of the records or to use a special end marker. Appending new records presents no problem; just use the command APPEND. Unfortunately, Apple does not have the EOF command. We shall simulate it with the command ONERR (= on error). The same goes for Commodore computers, which don't know EOF either.

When creating your own file, you must decide on one of these methods. Your decision depends both on the kind of computer you have and on the way you want to process your records. If, for instance, you don't want to append new records to the file, the second method will do nicely. We will stay with the third method for all the programs in this book.

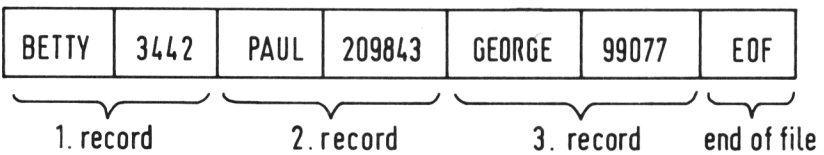


Fig.3.4; File End with EOF Label

The program PHONE, which we are going to develop in this chapter, is a modular construction. First of all, the main program is composed, and then you can write and test the individual subroutines one by one. Don't think of the program as a finished product capable of all kinds of things, but rather as a basic framework which shows you the principles of file processing, and which we hope will encourage you to expand and improve it.

N.B.: If you use the ONERR command in a subroutine on Apple, the GOSUB-RETURN mechanism will go haywire. As far as possible, you should use a GOTO command instead of the RETURN command. While the program is running, the error OUT OF MEMORY may occur (there is no more data storage room left). In this case, start the program again with RUN.

3.2 The Program Menu

Almost all programs offering a choice of processes work with a so-called menu. Figure 3.5 shows how we will use the **menu** technique. The individual processes offered by the menu are numbered 1 to K, whereby the last number K always signifies the end. The user chooses a number N, which must come between 1 and K; this will be subsequently checked. If the number is incorrect, a new input will be requested. With the command

```
ON N GOSUB 1000,2000,3000,....,9999
```

we branch off to the appropriate subroutine, and afterwards return to the menu. The case of N=K will be treated separately. The line 9999 contains the END command. For simplicity's sake, each subroutine begins with a round thousand number. If you stick to this pattern, your programs will be very clear.

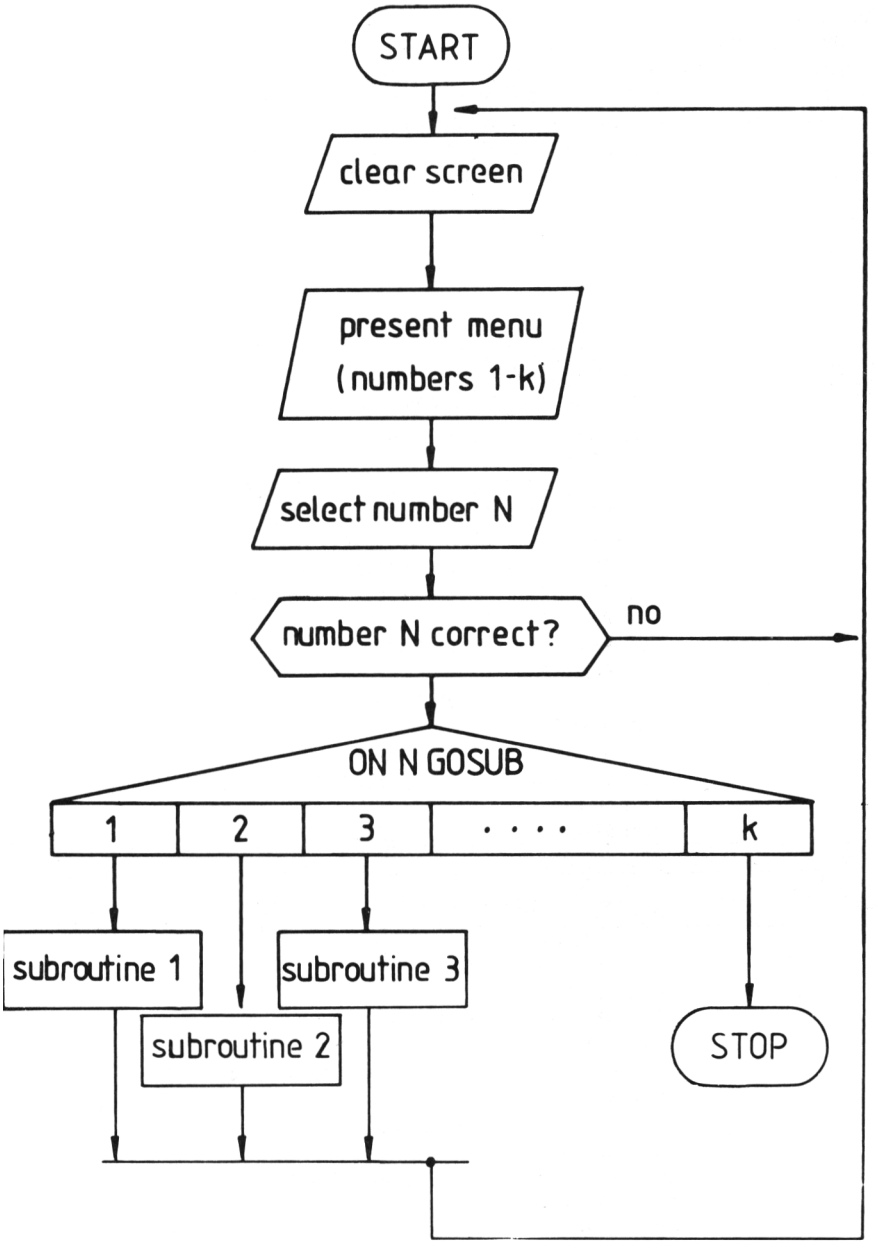


Fig.3.5; General pattern in menu techniques

There now follows the menu for the program PHONE. In line 20, the variable F$ is given as value the name PFILE (phone file); for Apple, the usual variable D$ is necessary. Clearing the screen is done by the subroutine from line 500 on. You should set your tabulator to the size of your screen. We always start from position 10. The six subroutines are each represented for the moment by a REM command, and will be added later. The advantage of this method is that you can expand your program in modules. In each of the following sections you will meet a new subroutine.

```
 10 REM=====PHONE=====
 20 LET F$="PFILE"
Apple:
 21 LET D$=CHR$(4)
 30 GOSUB 500: REM CLEAR SCREEN
 40 PRINT TAB(10);"PHONE DIRECTORY"
 50 PRINT:PRINT
 60 PRINT TAB(10);"1=CREATE DIRECTORY"
 70 PRINT TAB(10);"2=PRINT DIRECTORY"
 80 PRINT TAB(10);"3=ADD NEW NAME AND NUMBER"
 90 PRINT TAB(10);"4=SEARCH FOR NAME"
100 PRINT TAB(10);"5=DELETE NAME AND NUMBER"
110 PRINT TAB(10);"6=COPY DIRECTORY"
120 PRINT TAB(10);"7=END"
130 PRINT:PRINT
140 PRINT TAB(10);"SELECT NUMBER,PLEASE"
150 INPUT N
160 IF N < 1 OR N > 7 THEN 140
170 ON N GOSUB 1000,2000,3000,4000,5000,6000,9999
180 GOTO 30
500 REM=====CLEAR SCREEN=====
Apple:
 510 HOME
MS,BBC:
 510 CLS
CBM:
 510 PRINT CHR$(147)
 520 RETURN
1000 REM=====CREATE PFILE=====
1010 RETURN
2000 REM=====READ AND PRINT PFILE=====
2010 RETURN
3000 REM=====ADD NEW NAMES AND NUMBERS=====
3010 RETURN
4000 REM=====SEARCH FOR NAME=====
4010 RETURN
5000 REM=====DELETE NAME AND NUMBER=====
5010 RETURN
6000 REM=====COPY PFILE TO PFILE2=====
6010 RETURN
9999 END
```

3.3 Creating the File PFILE

We will assume that the complete file will be created in one go, as there are relatively few names and numbers to deal with. Now and again some new names will later be added or old ones deleted.

Figure 3.6 shows the procedure. The file F$ is opened for reading. If you have already created a file with this name, you should delete this first, before selecting Point 1 from the menu. In line 1040 a name is entered. Here, as in all further programs in this book, we'll agree to end an input loop with three asterisks. These are not written to the file, however.

Enter the phone number in line 1060. This is not entered as a number but as **a string.** Thus you can also enter longer numbers. If you make a typing mistake, the whole input is deleted and you start at the beginning again.

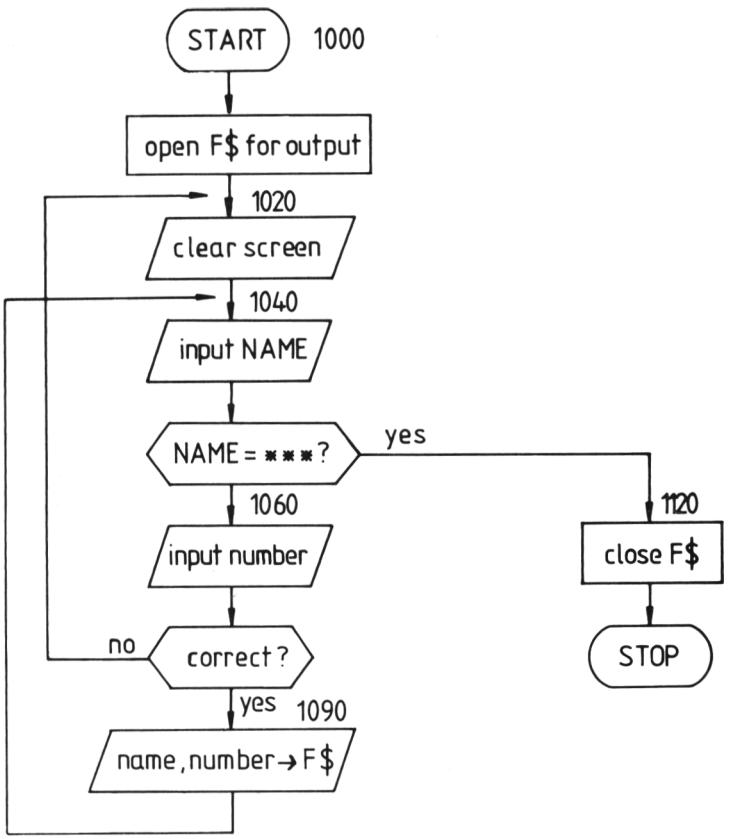


Fig. 3.6; flow chart; creating the file

Now let's look at the program. In the subroutine 1200 - 1220, the file PFILE is opened for writing. As the file name is contained in F$, you save a lot of writing with these commands. With Commodore, the file name contained in F$ is put before the letters S and W with the concatenation operator +. After name and phone number have been entered, both are written to the file (line 1090). With Apple, there are again those peculiarities to be looked out for which we met in the last chapter. The subroutine 1300 - 1320 closes the file when you have entered the three asterisks.

```
1000 REM=====CREATE FILE=====
1010 GOSUB 1200:REM OPEN PFILE FOR OUTPUT
1020 GOSUB 500
1030 PRINT"TYPE NAME AND PHONE NUMBER(***=STOP)":PRINT
1040 INPUT"NAME";N$
1050 IF N$="***"THEN 1120
1060 INPUT"PHONE NUMBER";P$
1070 INPUT"CORRECT(Y/N)";A$
1080 IF A$<>"Y"THEN 1020
```

Apple:
```
1090 PRINT D$;"WRITE",F$
1091 PRINT N$:PRINT P$
1092 PRINT D$
```

MS,CBM:
```
1090 PRINT#1,N$:PRINT#1,P$
```

BBC:
```
1090 PRINT#X,N$:PRINT#X,P$
```

```
1100 PRINT:PRINT
1110 GOTO 1030
1120 GOSUB 1300:REM CLOSE PFILE
1130 RETURN
1200 REM=====OPEN PFILE FOR OUTPUT=====
```

Apple:
```
1210 PRINT D$;"OPEN",F$
```

MS:
```
1210 OPEN F$ FOR OUTPUT AS#1
```

CBM:
```
1210 OPEN1,8,2F$+",S,W"
```

BBC:
```
1210 X=OPENOUT F$
```

```
1220 RETURN
1300 REM=====CLOSE PFILE=====
```

Apple:
```
1310 PRINT D$;"CLOSE",F$
```

MS,CBM:
```
1310 CLOSE 1
```

BBC:
```
1310 CLOSE#X
```

```
1320 RETURN
```

3.4 Reading and Printing from the File PFILE

Have a look at the flow chart in Figure 3.7. After the screen has been cleared, the file PFILE is opened for reading. Then the title is printed. If the end of the file has been reached, it will be closed. To enable you to retain the list on the screen, the question "CONTINUE?" appears. Only with a positive answer will you return to the menu.

Name and number are read from the file at line 2060 and then printed out.

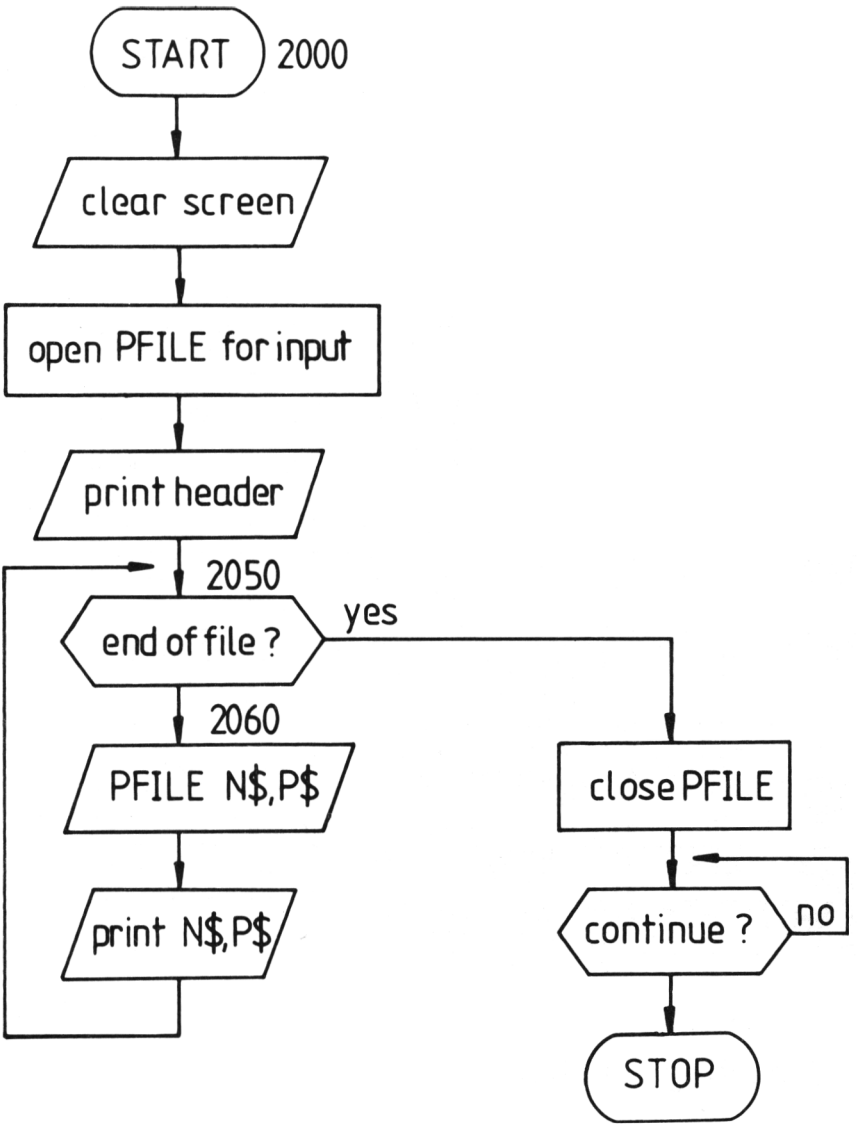


Fig. 3.7; flow chart; reading and printing from the file

The program allows for the fact that Apple and Commodore have no EOF command. With Apple, the ONERR command is used, and this has two consequences: in line 2089 the error code number must be set to 0. Further, the RETURN command in line 2130 must be replaced by a simple GOTO command, otherwise an error message will occur.

With the Commodore, the system variable ST is checked. If the end of the file has been reached, this takes on the value of 64. The other versions manage without this programming trick.

```
                2000 REM=====READ AND PRINT FILE=====
                2010 GOSUB 500
                2020 GOSUB 2200:REM OPEN PFILE FOR INPUT
Apple:          2021 PRINT D$;"READ";F$
                2030 PRINT"LIST OF PHONE NUMBERS"
                2040 PRINT"--------------------"
Apple:          2050 ONERR GOTO 2089
MS:             2050 IF EOF(1) THEN 2090
CBM:            2050 IF ST=64 THEN 2090
BBC:            2050 IF EOF#X THEN 2090
Apple:          2060 INPUT N$,P$
MS,CBM:         2060 INPUT#1,N$,P$
BBC:            2060 INPUT#X,N$,P$
                2070 PRINT N$;TAB(20);P$
                2080 GOTO 2050
Apple:          2089 POKE 216,0
                2090 GOSUB 1300
                2100 PRINT:PRINT
                2110 INPUT"CONTINUE(Y/N)";A$
                2120 IF A$<>"Y"THEN 2110
Apple:          2130 GOTO 30
others:         2130 RETURN
                2200 REM=====OPEN PFILE FOR INPUT=====
Apple:          2210 PRINT D$;"OPEN";F$
MS:             2210 OPEN F$ FOR INPUT AS#1
CBM:            2210 OPEN 1,8,2,F$+",S,R"
BBC:            2210 X=OPENIN F$
                2220 RETURN
```

3.5 Adding new Records

Figure 3.8 shows the flow chart, which is very similar to that in Figure 3.6. The difference is that here the file is not opened for writing, but for adding (or appending) new records, to the end of the file (APPEND).

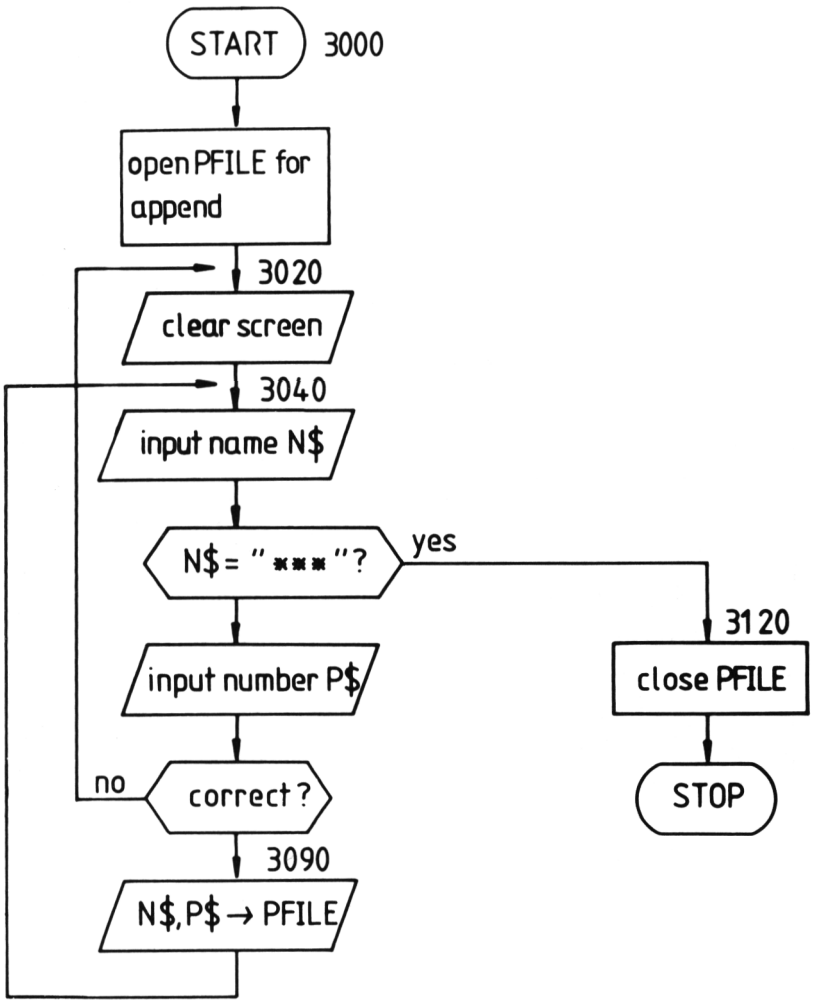


Fig.3.8; flow chart: appending new records

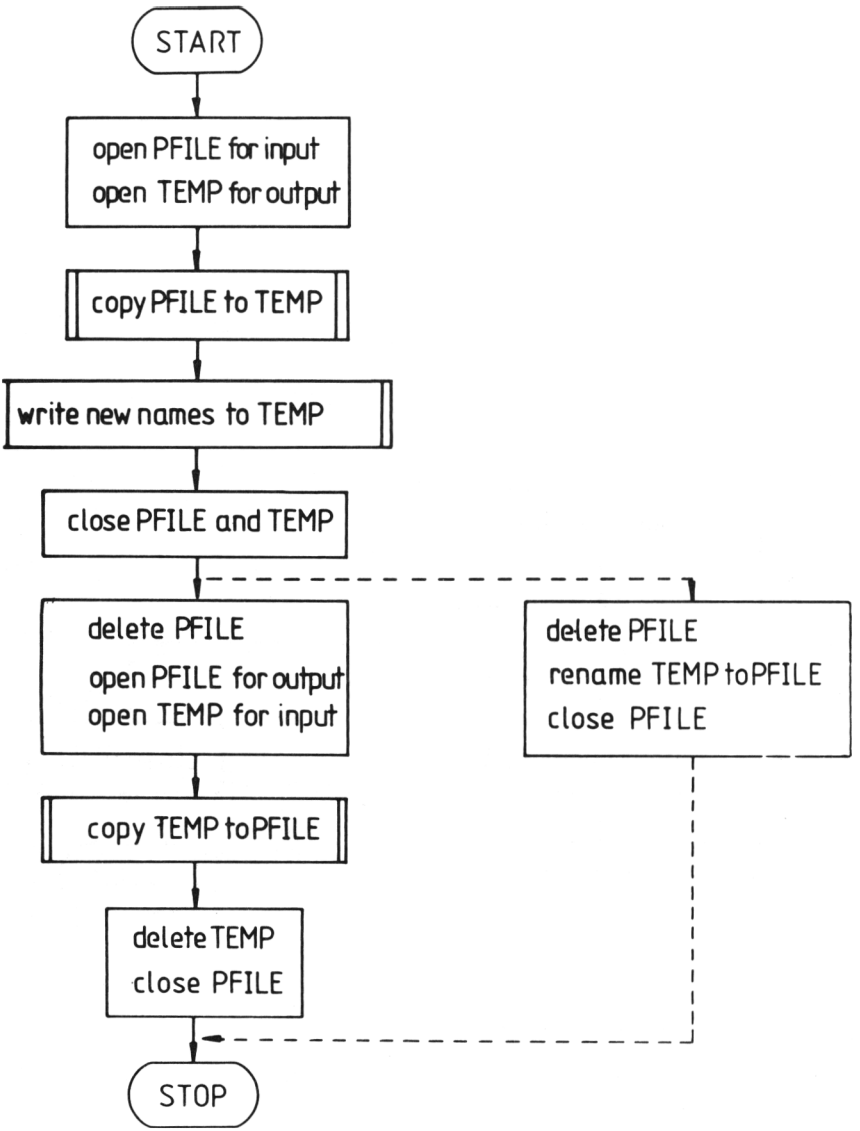


Fig. 3.9; flow chart: adding new records without the APPEND-mode

What can be done if your computer, like the BBC, for example, has no APPEND mode? In this case, you have to copy the whole file to an auxiliary file TEMP (for temporary) and add the new records to this. Then delete the original PFILE, open again for writing, and copy all the records from TEMP to PFILE. Figure 3.9 shows a flow chart for this. The technical details for the programming can be found in section 3.8, where there is a program for the copying of a file. With BBC, you can also go to the end of the file (EXT#X) using the command PTR#X and then add records. This possibility is shown in the following program.

If your computer knows the command RENAME (for giving the file a new name), you can save yourself the copying from TEMP to PFILE. Delete PFILE and simply give the name PFILE to the file TEMP. We will be discussing this method in more detail in Chapter 4; in the flow chart it is indicated by a dotted line.

```
3000 REM=====ADD NEW NAMES AND NUMBERS====
3010 GOSUB 3200:REM OPEN PFILE FOR APPEND
3020 GOSUB 500
3030 PRINT"TYPE NAME AND PHONE NUMBER(***=STOP)":PRINT
3040 INPUT"NAME";N$
3050 IF N$="***"THEN 3120
3060 INPUT"PHONE NUMBER";P$
3070 INPUT"CORRECT(Y/N)";A$
3080 IF A$<>"Y"THEN 3020
Apple:  3089 PRINT D$;"APPEND";F$
        3090 PRINT D$;"WRITE";F$
        3091 PRINT N$:PRINT P$
        3092 PRINT D$
MS,CBM: 3090 PRINT#1,N$:PRINT#1,P$
BBC:    3090 PRINT#X,N$:PRINT#X,P$
3100 PRINT:PRINT
3110 GOTO 3030
3120 GOSUB 1300
3130 RETURN
3200 REM=====OPEN PFILE FOR APPEND=====
Apple:  3210 PRINT D$;"OPEN";F$
MS:     3210 OPEN F$ FOR APPEND AS#1
CBM:    3210 OPEN 1,8,2,F$+",S,A"
BBC:    3210 X=OPENUP F$:PTR#X=EXT#X
3220 RETURN
```

3.6 Finding a Record

In Figure 3.10 you see the flow chart for the search routine. The name to be found is assigned to S$. Subsequently, all the names and phone numbers from the file PFILE are read and compared with S$. If there is no corresponding name, the message NAME NOT FOUND will appear when the end of the file is reached.

If the name is found, then it is displayed on the screen together with the phone number.

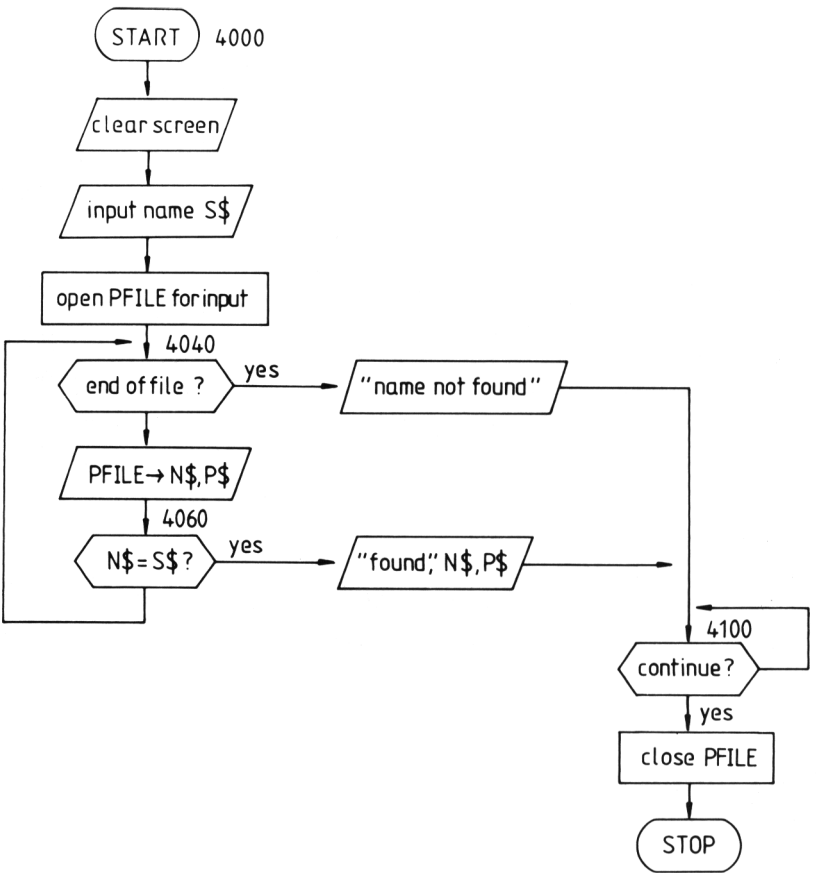


Fig.3.10; flow chart: finding a record

```
                4000 REM=====SEARCH FOR NAME=====
                4010 GOSUB 500
                4020 INPUT"NAME TO SEARCH FOR";S$
                4030 GOSUB 2200:REM OPEN PFILE FOR INPUT
Apple:          4031 PRINT D$;"READ";F$
                4040 ONERR GOTO 4089
MS:             4040 IF EOF(1)THEN 4090
CBM:            4040 IF ST=64 THEN 4090
BBC:            4040 IF EOF#X THEN 4090
Apple:          4050 INPUT N$,P$
MS,CBM:         4050 INPUT#1,N$,P$
BBC:            4050 INPUT#X,N$,P$
                4060 IF N$<>S$ THEN 4040
                4070 PRINT"FOUND:":PRINT N$,P$
Apple:          4071 PRINT D$
                4080 GOTO 4100
Apple:          4089 POKE 216,0
                4090 PRINT"NAME NOT FOUND"
                4100 INPUT"CONTINUE(Y/N)";A$
                4110 IF A$<>"Y"THEN 4100
                4120 GOSUB 1300:REM CLOSE PFILE
Others:         4130 RETURN
Apple:          4130 GOTO 30
```

3.7 Deleting a Record

To remove a record from the file PFILE, proceed as follows (see Figure 3.11). You open PFILE for reading and also an auxiliary file TEMP for writing. You read a record and compare it with the one to be deleted. If they are the same, you move on to the next record; if not, you write the record to the auxiliary file. Finally you either copy all the records from TEMP to PFILE, or give TEMP the name PFILE using the RENAME command.

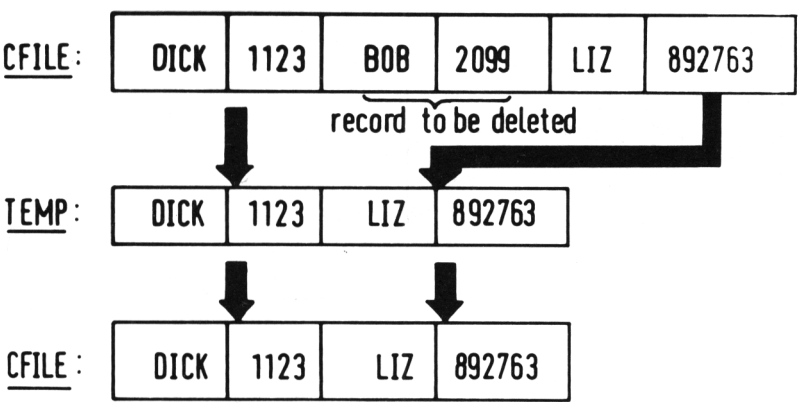


Fig.3.11; example of deletion of a record

Figure 3.12 shows the flow chart. The name to be deleted is assigned to the variable ND$. If you wish, you can have the message NAME NOT FOUND appear on the screen if the name is not contained in the file at all.

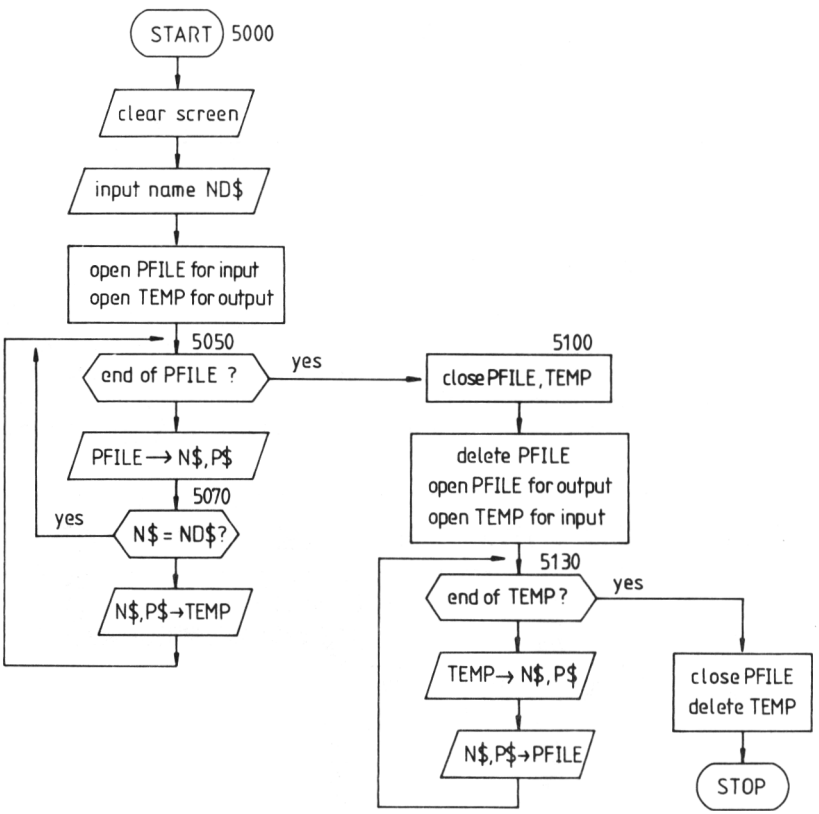


Fig.3.12; flow chart: deleting a record

The program shows a peculiarity of Commodore. As two files are opened at the same time, you must store the value of the system variable ST in an auxiliary variable TS after every reading, as the variable ST is given a new value again in the subsequent writing. This variable TS is checked in lines 5050 and 5130. On entry into the read loop it is given the value 0.

The program shows in detail how to copy the file TEMP back to the original file PFILE. From now on we'll avoid this rather laborious method and work with the RENAME command instead. Notice that the BBC computer requires an extra little subroutine (10000) and this is described at the end of this chapter.

```
5000 REM=====DELETE NAME AND NUMBER=====
5010 GOSUB 500
5020 INPUT"NAME TO DELETE";ND$
5030 GOSUB 2200:REM OPEN PFILE FOR INPUT
5040 GOSUB 5300:REM OPEN TEMP FOR OUTPUT
CBM:    5041 LET TS=0
Apple:  5050 PRINT D$;"READ";F$
        5051 ONERR GOTO 5099
MS:     5050 IF EOF(1)THEN 5100
CBM:    5050 IF TS=64 THEN 5100
BBC:    5050 IF EOF#X THEN 5100
Apple:  5060 INPUT N$,P$
        5061 PRINT D$
MS,CBM: 5060 INPUT#1,N$,P$
CBM:    5061 LET TS=ST
BBC:    5060 INPUT#X,N$,P$
        5070 IF N$=ND$ THEN 5050
Apple:  5080 PRINT D$;"WRITE";"TEMP"
        5081 PRINT N$:PRINT P$:PRINT D$
MS,CBM: 5080 PRINT#2,N$:PRINT#2,P$
BBC:    5080 PRINT#Y,N$:PRINT#Y,P$
        5090 GOTO 5050
Apple:  5099 POKE 216,0
        5100 GOSUB 5400:REM CLOSE TEMP
        5110 GOSUB 5600:REM DELETE PFILE AND OPEN FOR OUTPUT
        5120 GOSUB 5500:REM OPEN TEMP FOR INPUT
CBM:    5121 LET TS=0
Apple:  5130 PRINT D$;"READ";"TEMP"
        5131 ONERR GOTO 5169
MS:     5130 IF EOF(2) THEN 5170
CBM:    5130 IF TS=64 THEN 5170
BBC:    5130 IF EOF#Y THEN 5170
Apple:  5140 INPUT N$,P$
        5141 PRINT D$
MS,CBM: 5140 INPUT#2,N$,P$
CBM:    5141 LET TS=ST
BBC:    5140 INPUT#Y,N$,P$
```

```
Apple:      5150 PRINT D$;"WRITE";F$
            5151 PRINT N$:PRINT P$:PRINT D$
MS,CBM:     5150 PRINT#1,N$:PRINT#1,P$
BBC:        5150 PRINT#X,N$:PRINT#X,P$
            5160 GOTO 5130
Apple:      5169 POKE 216,0
            5170 GOSUB 1300:REM CLOSE PFILE
            5180 GOSUB 5700:REM DELETE TEMP
Others:     5190 RETURN
Apple:      5190 GOTO 30

            5300 REM=====OPEN TEMP FOR OUTPUT=====
Apple:      5310 PRINT D$;"OPEN";"TEMP"
MS:         5310 OPEN "TEMP"FOR OUTPUT AS#2
CBM:        5310 OPEN 2,8,3,"TEMP,S,W"
BBC:        5310 Y=OPENOUT "TEMP"
            5320 RETURN

            5400 REM=====CLOSE TEMP=====
Apple:      5410 PRINT D$;"CLOSE";"TEMP"
MS,CBM:     5410 CLOSE 2
BBC:        5410 CLOSE#Y
            5420 RETURN

            5500 REM=====OPEN TEMP FOR INPUT=====
Apple:      5510 PRINT D$;"OPEN";"TEMP"
MS:         5510 OPEN"TEMP"FOR INPUT AS#2
CBM:        5510 OPEN 2,8,3,"TEMP,S,R"
BBC:        5510 Y=OPENIN "TEMP"
            5520 RETURN

            5600 REM=====DELETE PFILE AND OPEN FOR OUTPUT===
Apple:      5610 PRINT D$;"DELETE";F$
MS:         5610 CLOSE 1: KILL F$
CBM:        5610 CLOSE 1:OPEN 3,8,15,"S:"+F$:CLOSE 3
BBC:        5610 CLOSE#X:CLI$="DELETE "+F$:GOSUB10000
Apple:      5620 PRINT D$;"OPEN",$
MS:         5620 OPEN F$ FOR OUTPUT AS#1
CBM:        5620 OPEN 1,8,2,F$+",S,W"
BBC:        5620 X=OPENOUT F$
            5630 RETURN

            5700 REM=====DELETE TEMP=====
Apple:      5710 PRINT D$;"DELETE";"TEMP"
MS:         5710 CLOSE 2: KILL"TEMP"
CBM:        5710 CLOSE 2: OPEN 3,8,15,"S:TEMP": CLOSE 3
BBC:        5705 CLOSE#Y
            5710 *DELETE"TEMP"
            5720 RETURN
```

3.8 Copying the File PFILE

At the end of this chapter we will consider how to copy the file PFILE into a second one called PFILE2. This is important from the point of view of **data security**. If you have a second disk drive, you copy the file PFILE2 onto a second diskette, which can then be kept in a safe place.

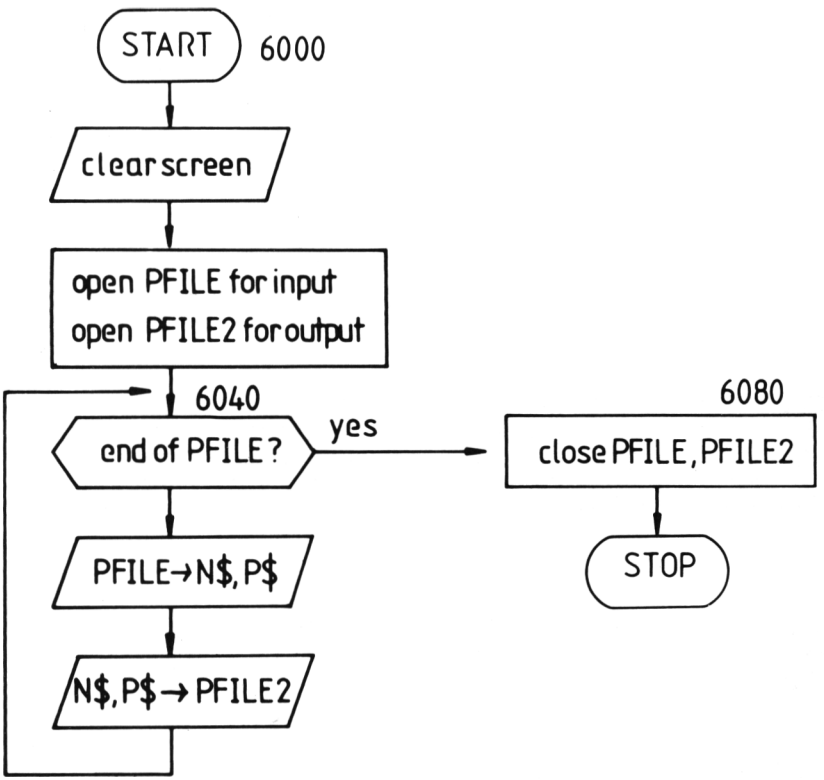


Figure 3.13 shows the flow chart. This is for a simple copying process, which continues to write records from PFILE to PFILE2 until it reaches the end of PFILE.

```
6000 REM=====COPY PFILE TO PFILE2=====
6010 GOSUB 500
6020 GOSUB 2200:REM OPEN PFILE FOR INPUT
6030 GOSUB 6200:REM OPEN PFILE2 FOR OUTPUT
CBM:    6031 LET TS=0
Apple:  6040 PRINT D$;"READ";F$
        6041 ONERR GOTO 6079
MS:     6040 IF EOF(1) THEN 6080
CBM:    6040 IF TS=64 THEN 6080
BBC:    6040 IF EOF#X THEN 6080
Apple:  6050 INPUT N$,P$
        6051 PRINT D$
MS,CBM: 6050 INPUT#1,N$,P$
CBM:    6051 LET TS=ST
BBC:    6050 INPUT#X,N$,P$
Apple:  6060 PRINT D$;"WRITE";"PFILE2"
        6061 PRINT N$:PRINT P$:PRINT D$
MS,CBM: 6060 PRINT#2,N$:PRINT#2,P$
BBC:    6060 PRINT#Y,N$:PRINT#Y,P$
        6070 GOTO 6040
Apple:  6079 POKE 216,0
        6080 GOSUB 1300:REM CLOSE PFILE
        6090 GOSUB 6300:REM CLOSE PFILE2
Others: 6100 RETURN
Apple:  6100 GOTO 30

        6200 REM=====OPEN PFILE2 FOR OUTPUT=====
Apple:  6210 PRINT  D$;"OPEN";PFILE2"
MS:     6210 OPEN"PFILE2"FOR OUTPUT AS#2
CBM:    6210 OPEN 2,8,3,"PFILE2,S,W"
BBC:    6210 Y=OPENOUT"PFILE2"
        6220 RETURN

        6300 REM=====CLOSE PFILE2====
Apple:  6310 PRINT D$;"CLOSE";"PFILE2"
MS,CBM: 6310 CLOSE 2
BBC:    6310 CLOSE#Y
        6320 RETURN
```

3.9 "BBC Subroutine 10000"

Under BASIC 1, you can not simply say "DELETE F$" as the name of the file is not substituted for F$. There is an easy way to do this on BASIC 2, but for both versions this subroutine, with line 5610, does the trick:

```
10000 REM-OSCLI
10010 $B%=CLI$:X%=B%MOD256:Y%=B%DIV256:CALL&FFF7
10020 RETURN
```

Chapter 4

Sorted Sequential Files

4.1 Example: A School Class List

A typical example of an alphabetically sorted file is the school class list. The surname, first name and the grades of each student are entered. We will restrict ourselves to three subjects:

MATH	- mathematics
PHYS	- physics
COMP	- computer science

The grades range from A to F. In Figure 4.1 there is an example of a file with two records.

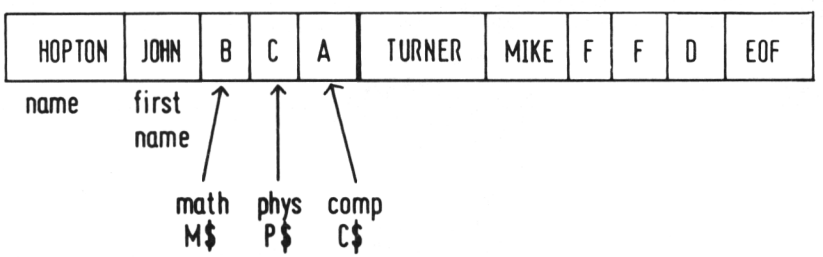


Fig.4.1; Example of a file with two records

We are going to design a program offering the teacher the following possibilities:

1. Creating a list
2. Printing a list
3. Adding new names
4. Finding names
5. Entering grades
6. Drawing up statistics
7. Deleting grades

We have deliberately left out the point "deleting names", as this subroutine is identical to the one in section 3.7, and wouldn't teach you anything new. Points 1,2 and 4 aren't essentially new, either, but they are too important to be left out.

Let us assume that the teacher is drawing up a new class list at the beginning of the school year. As yet, none of the students has been given any grades, so the surnames and first names are all that has to be entered. The grades are entered for the whole class in one process, as, for example, after a mathematics exam. If the teacher continues to teach the same class in the next school year, he needn't enter the names a second time; he merely deletes the grades.

Point 6,"Drawing up statistics", enables the teacher to examine, at any time, the standard of his class. For example, he can make a list of all students who have better grades than C or worse than D in mathematics; or the computer can print out the names of all students who have got a grade A in physics.

Of course, the program does not perform all the tasks occurring in school; it does, however, show you how to deal with sorted sequential files, and gives suggestions as to the development of similar programs for your own special needs.

NOTE:

If your computer displays less than 40 characters per line (e.g. VIC 20), you must change the TAB-commands accordingly.

There now follows the menu, which is very similar to the one in Chapter 3. The program is given the name CLASS. Let's call the file CFILE (class file).

```
                10 REM=====CLASS=====
BBC             12 DIM B%50, G1$(3)
                15 DIM NAM$(50,20),G$(50,3),NUM(6,3)
                20 LET F$="CFILE"
Apple:          21 LET D$=CHR$(4)
                30 GOSUB 500:REM CLEAR SCREEN
                40 PRINT TAB(10);"CLASS LIST"
                50 PRINT:PRINT
                60 PRINT TAB(10);"1=CREATE LIST"
                70 PRINT TAB(10);"2=PRINT LIST"
                80 PRINT TAB(10);"3=INSERT NEW NAME"
                90 PRINT TAB(10);"4=SEARCH FOR NAME"
               100 PRINT TAB(10);"5=ENTER GRADES"
               110 PRINT TAB(10);"6=STATISTICS"
               120 PRINT TAB(10);"7=DELETE ALL GRADES"
               130 PRINT TAB(10);"8=END"
               140 PRINT:PRINT
               150 PRINT TAB(10);"SELECT NUMBER,PLEASE"
               160 INPUT N
               170 IF N<1 OR N>8 THEN 150
               180 ON N GOSUB 1000,2000,3000,4000,5000,6000,
                   7000,9999
               190 GOTO 30
```

```
500 REM=====CLEAR SCREEN=====
Apple:  510 HOME
MS,BBC: 510 CLS
CBM:    510 PRINT CHR$(147)
520 RETURN
```

4.2 Creating the file CFILE

When creating the file, the surname and first name of each student is read in. The variables M$,P$ and C$, which contain the grades for mathematics, physics and computer science respectively, are each initialized with blanks. The subroutines have the same numbers as in Chapter 3, so you can take these over directly.

```
1000 REM=====CREATE CFILE=====
1010 LET M$=" ":LET P$=" ":LET C$=" "
1020 GOSUB 500
1030 GOSUB 1200:REM OPEN CFILE FOR OUTPUT
1040 PRINT"TYPE NAME AND FIRST NAME(***=STOP)"
1050 INPUT"NAME";N$
1060 IF N$="***"THEN 1130
1070 INPUT"FIRST NAME";V$
1080 INPUT"CORRECT(Y/N)";A$
1090 IF A$<>"Y" THEN 1050
Apple:  1100 PRINT D$;"WRITE";F$
        1101 PRINT N$:PRINT V$:PRINT M$:PRINT P$:PRINT C$
        1102 PRINT D$
MS,CBM: 1100 PRINT#1,N$:PRINT#1,V$
        1105 PRINT#1,M$:PRINT#1,P$:PRINT#1,C$
BBC:    1100 PRINT#X,N$:PRINT#X,V$
        1105 PRINT#X,M$:PRINT#X,P$:PRINT#X,C$
1110 PRINT:PRINT
1120 GOTO 1050
1130 GOSUB 1300:REM CLOSE CFILE
1140 RETURN

1200 REM=====OPEN CFILE FOR OUTPUT=====
Apple:  1210 PRINT D$;"OPEN";F$
MS:     1210 OPEN F$ FOR OUTPUT AS#1
CBM:    1210 OPEN 1,8,2,F$+",S,W"
BBC:    1210 X=OPENOUT F$
1220 RETURN

1300 REM=====CLOSE CFILE=====
Apple:  1310 PRINT D$;"CLOSE";F$
MS,CBM: 1310 CLOSE 1
BBC:    1310 CLOSE#X
1320 RETURN
```

4.3 Reading from and printing the file CFILE

The program is almost identical to that in section 3.4. Besides surname and first name, the three grades M$,P$ and C$ are read. Figure 4.2 shows a print-out.

SCHOOL CLASS LIST

NAME	FIRST NAME	MATH	PHYS	COMP
ARMSTRONG	MIKE	B	C	A
BAKER	CHRIS	F	C	B
BURTON	MARY	F	D	A
CHAMBERS	SAM	D	E	B
DANIELS	TED	A	D	A
KING	BARRY	D	E	C
MANSFIELD	JOHN	D	C	A
MAY	PETER	E	C	B
NEWMAN	JOHN	B	A	C
ROMFORD	LIZ	E	F	A
SADDLER	PEGGY	B	B	F
SPENCER	DEBORAH	A	B	D
STEWART	SALLY	D	C	E
TEMPLETON	BILL	E	F	B
WALTHER	DAVID	C	E	C

Fig.4.2; example of a class list

```
2000 REM=====READ AND PRINT CFILE=====
2010 GOSUB 500
2020 GOSUB 2200:REM OPEN CFILE FOR INPUT
Apple: 2021 PRINT D$,"READ",F$
2030 PRINT "SCHOOL CLASS LIST"
2040 PRINT "================="
2050 PRINT "NAME";TAB(14);"FIRST NAME"
2060 PRINT TAB(25);"MATH";TAB(30);"PHYS";TAB(35);"COMP"
2070 PRINT "========================="
Apple: 2080 ONERR GOTO 2129
MS: 2080 IF EOF(1) THEN 2130
CBM: 2080 IF ST=64 THEN 2130
BBC: 2080 IF EOF#X THEN 2130
Apple: 2090 INPUT N$,V$,M$,P$,C$
MS,CBM: 2090 INPUT#1,N$,V$,M$,P$,C$
```

```
BBC:    2090 INPUT#X,N$,V$,M$,P$,C$
        2100 PRINT N$;TAB(14);V$;
        2110 PRINT TAB(27);M$;TAB(32);P$;TAB(37);C$
        2120 GOTO 2080
Apple:  2129 POKE 216,0
        2130 GOSUB 1300:REM CLOSE CFILE
        2140 PRINT:PRINT
        2150 INPUT"CONTINUE(Y/N)";A$
        2160 IF A$<>"Y"THEN 2150
Apple:  2170 GOTO 30
Others: 2170 RETURN

        2200 REM=====OPEN CFILE FOR INPUT=====
Apple:  2210 PRINT D$;"OPEN",F$
MS:     2210 OPEN F$ FOR INPUT AS#1
CBM:    2210 OPEN 1,8,2,F$+",S,R"
BBC:    2210 X=OPENIN F$
        2220 RETURN
```

4.4 Inserting a new record

Inserting into a sorted file does not occur at the end but in the right place, determined by the alphabetical order. The algorithm is a little more complicated; let's look at it in Figure 4.3.

First of all, the new record is read in (line 3020). The file CFILE is opened for reading, the file TEMP for writing. In a loop from 3110 to 3130, the records are copied from CFILE to TEMP for as long as the new name is alphabetically "smaller" than the name being read. If this is not the case (line 3140), then first the new record is written to TEMP, then the old one. Subsequently, the remaining records are copied from CFILE to TEMP. Instead of copying the whole file back, we simply give it a new name using the RENAME command (line 3250): the file CFILE is deleted, the file TEMP is given the name CFILE. In the case of longer files, a lot of time is saved in this way.

By means of examples you will clearly see that the algorithm shown in Figure 4.3 also functions in the two special cases of inserting the new record at the very beginning or end of the file. Inserting at the beginning happens in line 3140, as the new name is alphabetically smaller than the first name read. Inserting at the end happens in line 3220. You get to this position when the whole file is read and all the names were alphabetically smaller than or equal to the new name.

```
3000 REM=====INSERT NEW NAME=====
3010 GOSUB 500
3020 INPUT"NAME";N1$
3030 INPUT"FIRST NAME";V1$
3040 INPUT"MATH";M1$
3050 INPUT"PHYS";P1$
3060 INPUT"COMP";C1$
```

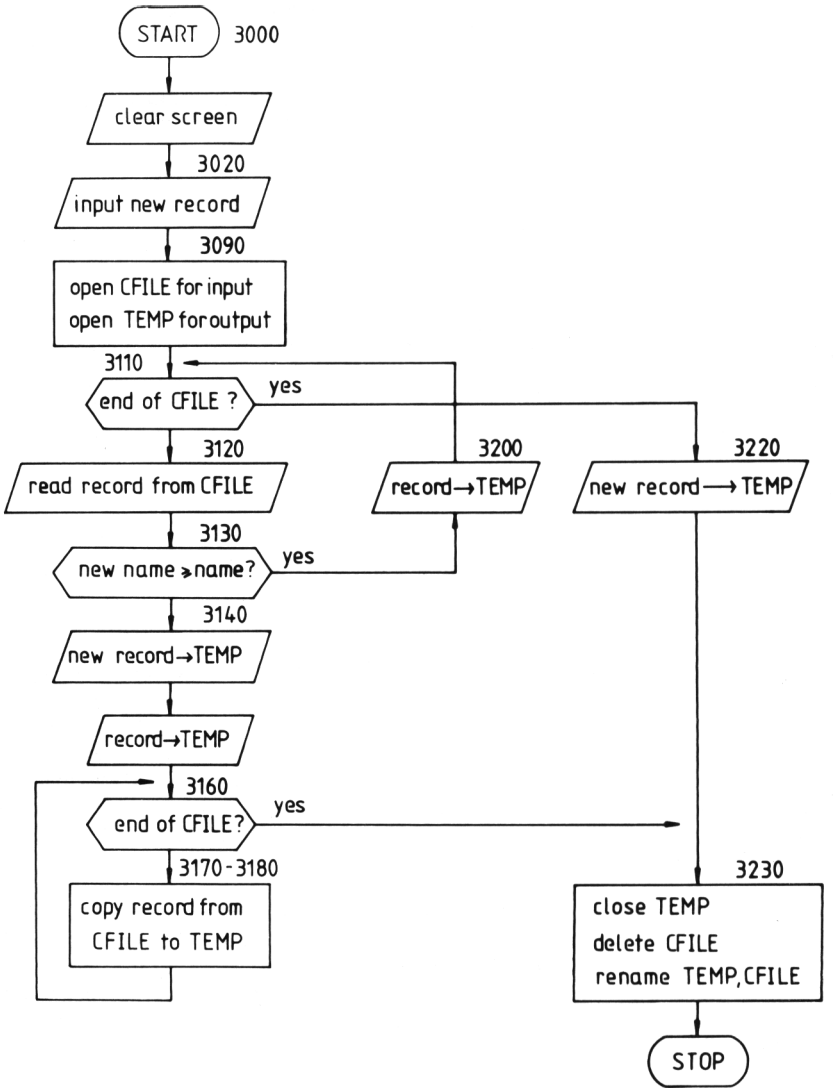


Fig.4.3; Flow chart: Inserting into a sorted file

```
3070 INPUT"CORRECT(Y/N)";A$
3080 IF A$<>"Y"THEN 3010
3090 GOSUB 2200:REM OPEN CFILE FOR INPUT
3100 GOSUB 3400:REM OPEN TEMP FOR OUTPUT
CBM:      3101 LET TS=0
Apple:    3110 PRINT D$;"READ";F$
          3111 ONERR GOTO 3218
MS:       3110 IF EOF(1)THEN 3220
CBM:      3110 IF TS=64 THEN 3220
BBC:      3110 IF EOF#X THEN 3220
Apple:    3120 INPUT N$,V$,M$,P$,C$
          3121 PRINT D$
MS,CBM:   3120 INPUT#1,N$,V$,M$,P$,C$
CBM:      3121 LET TS=ST
BBC:      3120 INPUT#X,N$,V$,M$,P$,C$
          3130 IF N$<=N1$ THEN 3200
Apple:    3140 PRINT D$;"WRITE","TEMP"
          3141 PRINT N1$:PRINT V1$:PRINT M1$:PRINT P1$:PRINT
               C1$
MS,CBM:   3140 PRINT#2,N1$:PRINT#2,V1$:PRINT#2,M1$:PRINT#2,
               P1$:PRINT#2,C1$
BBC:      3140 PRINT#Y,N1$:PRINT#Y,V1$:PRINT#Y,M1$:PRINT#Y,
               P1$:PRINT#Y,C1$
Apple:    3150 PRINT N$:PRINT V$:PRINT M$:PRINT P$:PRINT C$
          3151 PRINT D$
MS,CBM:   3150 PRINT#2,N$:PRINT#2,V$:PRINT#2,M$:PRINT#2,P$:
               PRINT#2,C$
BBC:      3150 PRINT#Y,N$:PRINT#Y,V$:PRINT#Y,M$:PRINT#Y,P$:
               PRINT#Y,C$
Apple:    3160 PRINT D$;"READ";F$
          3161 ONERR GOTO 3229
MS:       3160 IF EOF(1) THEN 3230
CBM:      3160 IF TS=64 THEN 3230
BBC:      3160 IF EOF#X THEN 3230
Apple:    3170 INPUT N$,V$,M$,P$,C$
          3171 PRINT D$
MS,CBM:   3170 INPUT#1,N$,V$,M$,P$,C$
CBM:      3171 LET TS=ST
BBC:      3170 INPUT#X,N$,V$,M$,P$,C$
Apple:    3180 PRINT D$;"WRITE","TEMP"
          3181 PRINT N$:PRINT V$:PRINT M$:PRINT P$:PRINT C$
          3182 PRINT D$
MS,CBM:   3180 PRINT#2,N$:PRINT#2,V$:PRINT#2,M$:PRINT#2,P$:
               PRINT#2,C$
BBC:      3180 PRINT#Y,N$:PRINT#Y,V$:PRINT#Y,M$:PRINT#Y,P$:
               PRINT#Y,C$
          3190 GOTO 3160
Apple:    3200 PRINT D$; "WRITE","TEMP"
          3201 PRINT N$:PRINT V$:PRINT M$:PRINT P$:PRINT C$
```

```
Apple:      3202 PRINT D$
```

```
MS,CBM:     3200 PRINT#2,N$:PRINT#2,V$:PRINT#2,M$:PRINT#2,P$:
                 PRINT#2,C$
BBC:        3200 PRINT#Y,N$:PRINT#Y,V$:PRINT#Y,M$:PRINT#Y,P$:
                 PRINT#Y,C$
            3210 GOTO 3110
Apple:      3218 POKE 216,0
            3219 PRINT D$:PRINT D$;"WRITE";"TEMP"
            3220 PRINT N1$:PRINT V1$:PRINT M1$:PRINT P1$:PRINT
                 C1$
            3221 PRINT D$
MS,CBM:     3220 PRINT#2,N1$:PRINT#2,V1$:PRINT#2,M1$:PRINT#2,
                 P1$:PRINT#2,C1$
BBC:        3220 PRINT#Y,N1$:PRINT#Y,V1$:PRINT#Y,M1$:PRINT#Y,
                 P1$:PRINT#Y,C1$
Apple:      3229 POKE 216,0
            3230 GOSUB 3500:GOSUB 1300:REM CLOSE
Apple:      3240 PRINT D$;"DELETE";F$
MS:         3240 KILL F$
CBM:        3240 OPEN 1,8,15,"S:"+F$
BBC:        3240 CLI$="DELETE "+F$:GOSUB 10000
Apple:      3250 PRINT D$;"RENAME TEMP,"F$
MS:         3250 NAME "TEMP" AS F$
CBM:        3250 PRINT#1,"R:CFILE=TEMP":CLOSE 1
BBC:        3250 CLI$="RENAME TEMP CFILE";GOSUB 10000
Apple:      3260 GOTO 30
Others:     3260 RETURN
            3400 REM=====OPEN TEMP FOR OUTPUT=====
Apple:      3410 PRINT D$;"OPEN";"TEMP"
MS:         3410 OPEN"TEMP"FOR OUTPUT AS#2
CBM:        3410 OPEN 2,8,3,"TEMP,S,W"
BBC:        3410 Y=OPENOUT"TEMP"
            3420 RETURN
            3500 REM=====CLOSE TEMP=====
Apple:      3510 PRINT D$;"CLOSE";"TEMP"
MS,CBM:     3510 CLOSE 2
BBC:        3510 CLOSE#Y
            3520 RETURN
```

NOTE: Subroutine 10000 is the same as the one listed on page 42.

4.5 Finding a record

Searching the file is done as described in section 3.6. As the line numbers also begin with 4000, you can in fact adopt parts of the earlier program.

```
                4000 REM=====SEARCH FOR NAME=====
                4010 GOSUB 500
                4020 INPUT"NAME TO SEARCH FOR";S$
                4030 GOSUB 2200:REM OPEN CFILE FOR INPUT
Apple:          4031 PRINT D$;"READ";F$
                4040 ONERR GOTO 4099
MS:             4040 IF EOF(1) THEN 4100
CBM:            4040 IF ST=64 THEN 4100
BBC:            4040 IF EOF#X THEN 4100
Apple:          4050 INPUT N$,V$,M$,P$,C$
MS,CBM:         4050 INPUT#1,N$,V$,M$,P$,C$
BBC:            4050 INPUT#X,N$,V$,M$,P$,C$
                4060 IF N$ <> S$ THEN 4040
                4070 PRINT"FOUND":PRINT N$" , ";V$;
                4080 PRINT TAB(30);M$;" ";P$;" ";C$
Apple:          4081 PRINT D$
                4090 GOTO 4110
Apple:          4099 POKE 216,0
                4100 PRINT"NAME NOT FOUND"
                4110 INPUT"CONTINUE(Y/N)";A$
                4120 IF A$ <> "Y" THEN 4110
                4130 GOSUB 1300:REM CLOSE CFILE
Others:         4140 RETURN
Apple:          4140 GOTO 30
```

Please note the following: the file CFILE is alphabetically sorted, but as you can only gain access to it sequentially, you must nevertheless do this one record at a time. A direct-access file allows a much faster search process.

4.6 Entering the grades

Designing this subroutine, we have assumed that the grades in each subject will be entered for all students. Let's say an **all students** physics exam has been held; the teacher enters the physics grade for each student. You can easily imagine what a supplementary program for entering the grade of one single student would look like.

Entering the grades - as with deleting all grades (see section 4.8) - belongs to the category "Altering a record". There are two possible ways of doing this.

1st Method: with auxiliary file

You read each record from CFILE into the workspace, and alter it there if necessary. Then you write it to the file TEMP. Finally, you either copy this file back to CFILE, or rename it.

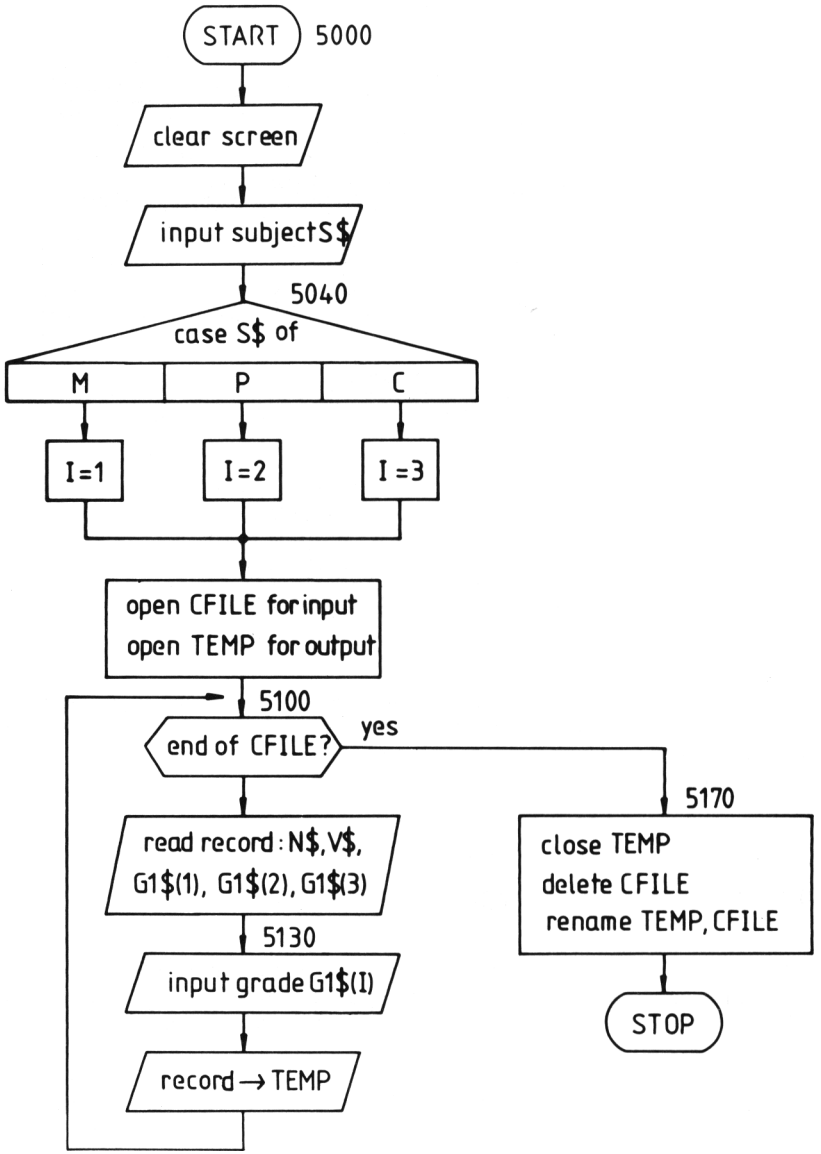


Fig.4.4; flow chart: Entering the grades

2nd Method: without auxiliary file

You read the record to be altered from CFILE into the workspace and alter it there; then you write it back to CFILE straightaway.

Your choice of method depends first of all on the kind of storage medium the file is on. If you were working with magnetic tape or cassette, you wouldn't be able to use the second method, as it is not possible to position the tape back. As we are using diskettes, it is in principle a usable method. However, writing to and reading from a sequential file simultaneously can lead to error in the case of some computers. For this reason we will use the first method. Figure 4.4 shows the flow chart.

The variable I takes on the value 1,2 or 3, according to whether mathematics, physics or computer science was chosen. The three grades of each student are read into an array G1$. Figure 4.5 shows an example.

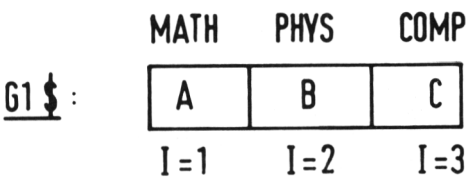


Fig.4.5; Example of occupation of array G1$

In line 5130 the new grade is entered and assigned to the 1st,2nd or 3rd element of array G1$. The altered record is written to TEMP. When the end of file CFILE is reached in line 5100, we rename it as usual.

```
5000 =====REM ENTER GRADES=====
5010 GOSUB 500
5020 PRINT"ENTER M FOR MATH,P FOR PHYS,C FOR
     COMP"
5030 INPUT S$
5040 IF S$ <>"M" AND S$<>"P"AND S$<>"C"THEN
     5010
5050 IF S$="M"THEN LET I=1:GOTO 5080
5060 IF S$="P"THEN LET I=2:GOTO 5080
5070 LET I=3
5080 GOSUB 2200:REM OPEN CFILE FOR INPUT
5090 GOSUB 3400:REM OPEN TEMP FOR OUTPUT
```

CBM:	5091 LET TS=0
Apple:	5100 PRINT D$;"READ";F$
	5101 ONERR GOTO 5169
MS:	5100 IF EOF(1) THEN 5170
CBM:	5100 IF TS=64 THEN 5170
BBC:	5100 IF EOF#X THEN 5170

```
Apple:          5110 INPUT N$,V$,G1$(1),G1$(2),G1$(3)
                5111 PRINT D$
MS,CBM:         5110 INPUT#1,N$,V$,G1$(1),G1$(2),G1$(3)
BBC:            5110 INPUT#X,N$,V$,G1$(1),G1$(2),G1$(3)
CBM:            5111 LET TS=ST
                5120 PRINT N$;" , ";V$
                5130 INPUT"GRADE";G1$(I)
                5140 PRINT:PRINT
Apple:          5150 PRINT D$;"WRITE";"TEMP"
                5151 PRINT N$:PRINT V$:PRINT G1$(1):PRINT G1$(2):
                     PRINT G1$(3)
                5152 PRINT D$
MS,CBM:         5150 PRINT#2,N$:PRINT#2,V$:PRINT#2,G1$(1):PRINT#2,
                     G1$(2):PRINT#2, G1$(3)
BBC:            5150 PRINT#Y,N$:PRINT#Y,V$:PRINT#Y,G1$(1):PRINT#Y,
                     G1$(2):PRINT#Y, G1$(3)
                5160 GOTO 5100
Apple:          5169 POKE 216,0
                5170 GOSUB 3500:REM CLOSE TEMP
                5180 GOSUB 1300:REM CLOSE CFILE
Apple:          5190 PRINT D$;"DELETE";F$
                5191 PRINT D$;"RENAME TEMP," F$
MS:             5190 KILL F$
                5191 NAME"TEMP"AS F$
CBM:            5190 OPEN 1,8,15,"S:"+F$
                5191 PRINT#1,"R:CFILE=TEMP"
                5192 CLOSE 1
BBC:            5190 CLOSE#X:CLI$="DELETE "+F$,GOSUB 10000
                5191 CLI$="RENAME TEMP CFILE":GOSUB10000
Others:         5200 RETURN
Apple:          5200 GOTO 30
```

4.7 Drawing up statistics

If you adapt a conventional card index to electronic data processing, not only does the work become simpler, quicker and more accurate, but also a good deal more statistical information is placed at your disposal. Let us design a statistics subroutine which, for example, answers the following questions from the teacher:

a) Which students have a better grade than C in physics?

b) Which students have a worse grade than D in mathematics?

c) Which students have the grade A in computer science?

d) How often does each grade occur in percent?

Answering such questions with a large conventional card index will in most cases take a lot of time. If the information is contained in a sequential file, however, then the computer can provide answers very quickly. The advantage of a sorted file lies in the fact that results are stored in alphabetical order and no longer need to be sorted.

The statistics subroutine offers its own menu. This technique is often used in professional software. You choose one point in the main menu and gain access to a submenu. From here, you can often gain access to further submenus.

Figure 4.6 shows two arrays, NAM$ and G$. The arrya NAM$ contains 50x2=100 elements for the storing of a maximum of 50 surnames and first names. The array G$ holds the grades, and consists of 50x3=150 elements. The number 50 is chosen arbitrarily. At the beginning, we read the records from CFILE to these arrays. The statistical evaluation is achieved by means of the data contained in these arrays. This helps us achieve a rapid evaluation, as the records only have to be read from the diskette once, and then can at once be analysed according to the various criteria as often as you wish.

NAM$

	1	2
1	ARMSTRONG	MIKE
2	BAKER	CHRIS
3	BURTON	MARY
	⋮	
50		

G$

	1	2	3
1	B	C	A
2	F	C	B
3	F	D	A
		⋮	
50			

Fig. 4.6; The arrays NAM$ and G$

In Figure 4.7 you can see the flow chart of the statistics main program. First of all the records are read and stored in the arrays NAM$ and G$. The variable N counts the number of records. Once all the records have been read, the menu is presented. Then follows a branch to the line numbers given. The first three subroutines are very short. The variable CHOICE$ is set to BETTER,WORSE or EQUAL, and then the common subroutine beginning in line 8400 is used.

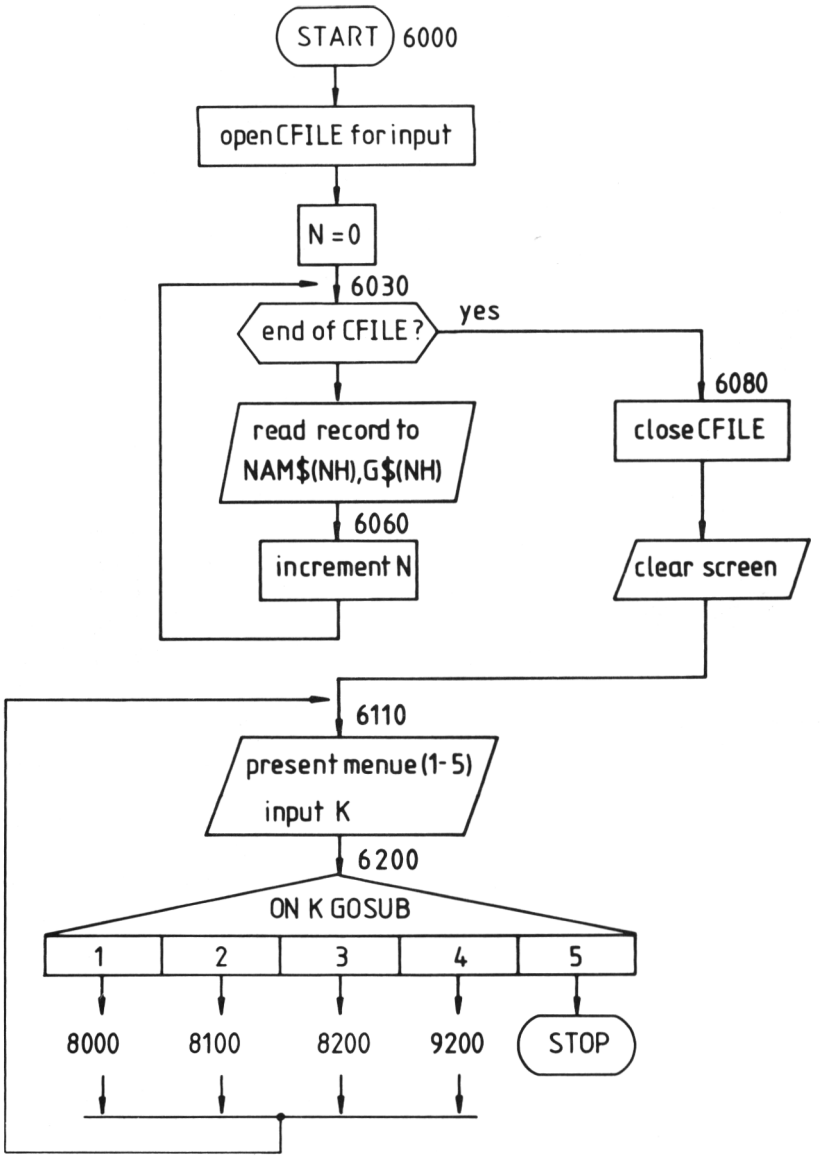


Fig. 4.7; flow chart: statistics menu

Let's now have a look at the individual statistics subroutines. Subject and grade are read in (lines 8420 to 8530). The variable S is given the following value according to subject:

S = 1 : MATH

S = 2 : PHYS

S = 3 : COMP

The counter COUNT is initialized by 0 in line 8540, and counts the number of students fulfilling the desired criterion (e.g., all those with the grade B in physics). In the FOR-loop from 8550 to 8610 the array G$ is examined. If the criterion is fulfilled, the name of the student is displayed. Finally, the total number is shown at the end of the list.

The subroutine for establishing an overall picture in percent begins in line 9200. It uses an array NUM, consisting of 6x3=18 elements. This array contains 18 counters for every grade and every subject. Each grade in the array G$ is examined, and causes the appropriate counter to be raised by one.

Firstly, all counters are initialized with zero (9220-9260). Then array G$ is read (9270-9320). Let's make ourselves familiar with the results of the commands in 9290 and 9300 by means of an example. Look at Figures 4.6 and 4.8.

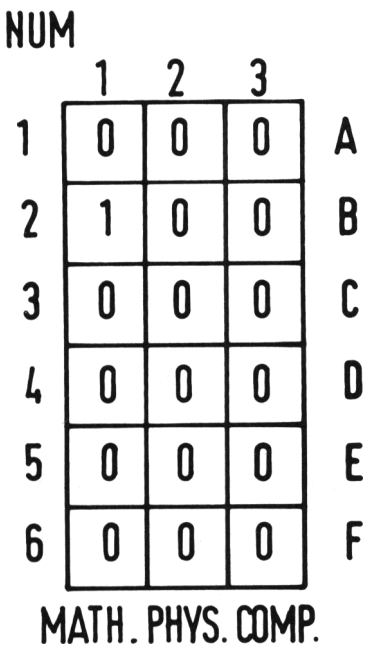


Fig. 4.8; The array NUM contains 18 counters

The variables I and J have at first both the value 1. In G$(1,1) there is the grade B, as student Armstrong has been given this grade in mathematics. So the variable P is given the following value:

```
9290 LET P=ASC("B")-64 =>P=66-64=2
```

Now the counter in the second line and first column of the array NUM is incremented:

```
9300 LET NUM(2,1)=NUM(2,1)+1
```

The other counters are raised in the same way. At the end, the percentages are calculated and displayed (9420-9430). We round them off to one decimal place. If your computer knows the command PRINT USING, you can arrange the display even better. Figure 4.9 shows an example print-out.

```
OVERALL PICTURE IN %
GRADE     MATH        PHYS        COMP
-------------------------------------------

  A       13.3        6.7         33.3
  B       20          13.3        26.7
  C       6.7         33.3        20
  D       26.7        13.3        6.7
  E       20          20          6.7
  F       13.3        13.3        6.7
```

Fig.4.9; Example of an overall picture in percent

```
                6000 REM===== STATISTICS=====
                6010 GOSUB 2200:REM OPEN CFILE FOR INPUT
Apple:          6011 PRINT D$;"READ";F$
                6020 LET N=0
Apple:          6030 ONERR GOTO 6079
MS:             6030 IF EOF(1)THEN 6080
CBM:            6030 IF ST=64 THEN 6080
BBC:            6030 IF EOF#X THEN 6080
Apple:          6040 INPUT NAM$(N+1,1),NAM$(N+1,2)
                6050 INPUT G$(N+1,1),G$(N+1,2),G$(N+1,3)
MS,CBM:         6040 INPUT#1,NAM$(N+1,1),NAM$(N+1,2)
                6050 INPUT#1,G$(N+1,1),G$(N+1,2),G$(N+1,3)
BBC:            6040 INPUT#X,NAM$(N+1,1),NAM$(N+1,2)
                6050 INPUT#X,G$(N+1,1),G$(N+1,2),G$(N+1,3)
                6060 LET N=N+1
                6070 GOTO 6030
Apple:          6079 POKE 216,0
                6080 GOSUB 1300:REM CLOSE CFILE
```

```
6090 GOSUB 500
6100 REM MENU
6110 PRINT TAB(10);"1=BETTER THAN"
6120 PRINT TAB(10);"2=WORSE THAN"
6130 PRINT TAB(10);"3=EQUAL TO"
6140 PRINT TAB(10);"4=OVERALL PICTURE(%)"
6150 PRINT TAB(10);"5=END"
6160 PRINT:PRINT
6170 PRINT TAB(10);"SELECT NUMBER,PLEASE"
6180 INPUT K
6190 IF K<1 OR K>5 THEN 6180
6200 ON K GOSUB 8000,8100,8200,9200,9500
```

Apple:	6210 GOTO 30
Others:	6210 RETURN

```
8000 REM=====BETTER THAN=====
8010 LET CHOICE$="BETTER"
8020 GOSUB 8400
8030 RETURN

8100 REM=====WORSE THAN=====
8110 LET CHOICE$="WORSE"
8120 GOSUB 8400
8130 RETURN

8200 REM=====EQUAL=====
8210 LET CHOICE$="EQUAL"
8220 GOSUB 8400
8230 RETURN

8400 REM=====SUBROUTINE FOR BETTER,WORSE,
     EQUAL=====
8410 GOSUB 500
8420 PRINT"SELECT SUBJECT:M=MATH,P=PHYS,C=COMP"
8430 INPUT S$
8440 IF S$<>"M"AND S$<>"P"AND S$<>"C"THEN
     8420
8450 IF S$="M"THEN LET S=1:GOTO 8480
8460 IF S$="P"THEN LET S=2:GOTO 8480
8470 LET S=3
8480 IF CHOICE$="EQUAL"THEN 8510
8490 PRINT CHOICE$;" THAN "
8500 INPUT B$:GOTO 8520
8510 INPUT"EQUAL TO ";B$
8520 IF B$<"A"OR B$>"F"THEN 8480
8530 PRINT:PRINT
8540 LET COUNT=0
8550 FOR I=1 TO N
8560 IF CHOICE$="BETTER"AND G$(I,S)>=B$ THEN 8610
```

```
8570 IF CHOICE$="WORSE" AND G$(I,S)<=B$ THEN
     8610
8580 IF CHOICE$="EQUAL" AND G$(I,S)<>B$ THEN
     8610
8590 PRINT NAM$(I,1);" , ";NAM$(I,2)
8600 LET COUNT=COUNT+1
8610 NEXT I
8620 PRINT:PRINT"TOTAL ";COUNT
8630 PRINT:PRINT
8640 INPUT"CONTINUE(Y/N) ";A$
8650 IF A$<>"Y"THEN 8640
8660 RETURN

9200 REM=====OVERALL PICTURE=====
9210 REM USES 18 COUNTERS NUM(6,3)
9220 FOR I=1 TO 6
9230 FOR J=1 TO 3
9240 LET NUM(I,J)=0
9250 NEXT J
9260 NEXT I
9270 FOR I=1 TO N
9280 FOR J=1 TO 3
9290 LET P=ASC(G$(I,J)) -64
9300 LET NUM(P,J)=NUM(P,J) +1
9310 NEXT J
9320 NEXT I
9330 GOSUB 500
9340 PRINT"OVERALL PICTURE IN %"
9350 PRINT"--------------------"
9360 PRINT"GRADE";TAB(8);"MATH";
9370 PRINT TAB(18);"PHYS";TAB(28);"COMP"
9380 PRINT "--------------------------"
9390 FOR I=1 TO 6
9400 PRINT TAB(3);CHR$(64+I);TAB(8);
9410 FOR J=1 TO 3
9420 LET PERC=NUM(I,J)*100/N
9430 PRINT(INT(PERC*10+0.5))/10;TAB(8+J*10);
9440 NEXT J
9450 PRINT
9460 NEXT I
9470 PRINT:PRINT
9480 INPUT"CONTINUE(Y/N)";A$
9490 IF A$<>"Y"THEN 9480
```

Apple:	9500 GOTO 30
Others:	9500 RETURN

```
9999 END
```

4.8 Deleting Grades

This subroutine deletes the grades of all the students in all subjects. It will be called up at the beginning of a new school year. In order to avoid the accidental deleting of all the grades - a disaster for the teacher, if perhaps a blessing for some students - , this part of the menu must be expressly confirmed once more (see line 7030).

```
7000 REM=====DELETE ALL GRADES=====
7010 GOSUB 500
7020 PRINT"ALL GRADES WILL BE DELETED"
7030 INPUT"TYPE Y TO CONFIRM";A$
7040 IF A$<>"Y"THEN RETURN
7050 GOSUB 2200:REM OPEN CFILE FOR INPUT
7060 GOSUB 3400:REM OPEN TEMP FOR OUTPUT
```

CBM:	7061 LET TS=0
Apple:	7070 PRINT D$;"READ";F$
	7071 ONERR GOTO 7109
MS:	7070 IF EOF(1)THEN 7110
CBM:	7070 IF TS=64 THEN 7110
BBC:	7070 IF EOF#X THEN 7110
Apple:	7080 INPUT N$,V$,M$,P$,C$
	7081 PRINT D$
MS,CBM:	7080 INPUT#1,N$,V$,M$,P$,C$
BBC:	7080 INPUT#X,N$,V$,M$,P$,C$
CBM:	7081 LET TS=ST
Apple:	7090 PRINT D$;"WRITE";"TEMP"
	7091 PRINT N$:PRINT V$:PRINT" ":PRINT" ":PRINT" "
	7092 PRINT D$
MS,CBM:	7090 PRINT#2,N$:PRINT#2,V$:PRINT#2," ":PRINT#2, " ": PRINT#2," "
BBC:	7090 PRINT#Y,N$:PRINT#Y,V$:PRINT#Y," ":PRINT#Y, " ": PRINT#Y," "
	7100 GOTO 7070

Apple:	7109 POKE 216,0
	7110 GOSUB 3500:GOSUB 1300:REM CLOSE
Apple:	7120 PRINT D$;"DELETE";F$
	7121 PRINT D$;"RENAME TEMP,"F$
MS:	7120 KILL F$
	7121 NAME"TEMP"AS F$
CBM:	7120 OPEN 1,8,15,"S:"+F$
	7121 PRINT#1,"R:CFILE=TEMP"
	7122 CLOSE 1
BBC:	7120 DELETE F$
	7121 RENAME"TEMP CFILE"
Others:	7130 RETURN
Apple:	7130 GOTO 30

Chapter 5

Random Access Files

5.1 Fundamentals

Sequential files have one great disadvantage: you cannot gain direct access to a particular record, it can only be reached indirectly. If, for example, you want to read the 17th record, you have to read the previous 16 first.

This disadvantage can be avoided by using **direct access files**, which are also called **random files** (files with optional access).

Random files cannot be realized on cassettes (magnetic tapes), but only on diskettes.

You can get a clear idea of the difference between direct and indirect access by considering a record player and a cassette recorder. If, for example, you want to hear the fourth piece of music on an LP, you can position the pickup arm in the appropriate place. With a cassette recorder, you first have to wind the tape forward. The LP, then, permits **rapid direct access** while, in the case of the cassette recorder, only a sequential, i.e. indirect access is possible.

The advantage of the rapid direct access with random files is, however, connected with a disadvantage: random files generally take up more room on the diskette. This is because **each record must have the same length.** Look at Figure 5.1 regarding this. It shows a file of three records, each consisting of a name. The sequential file only needs as much room as the length of the name demands. In the example this is:

ARMSTRONG ↵	10 bytes
KING ↵	5 bytes
NEWMAN ↵	7 bytes
total	22 bytes

With the random file you must use a fixed record length. For example, if you choose 10 bytes, the file needs 30 bytes for the three names, that is, 8 bytes more than the sequential file. You throw away, in effect, a considerable amount of storage space.

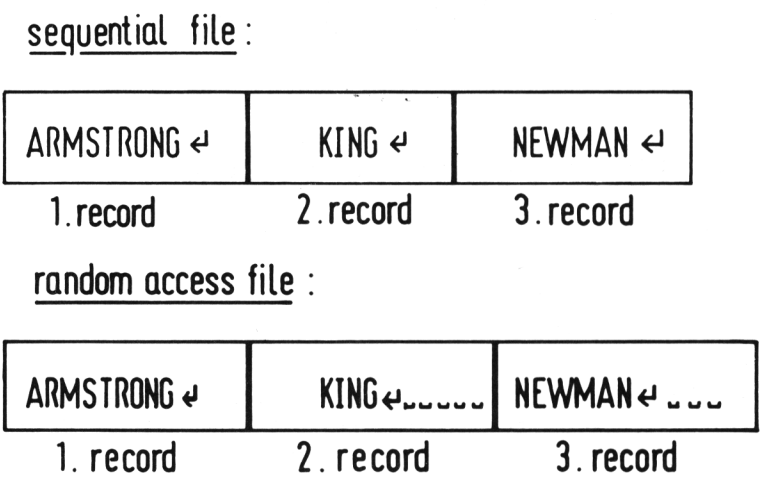


Fig. 5.1; comparison of room needed by a sequential and by a random file

The fixed record length is necessary for logical reasons. If you should want to read the third record, for example, you can calculate the address of the first byte of this record relative to the beginning of the file as follows:

(3 - 1) x length of record = 2 x 10 = 20

We will assume that the numbering of the bytes begins at 0. Thus the first record occupies bytes 0 to 9, the second bytes 10 to 19. The third record starts at byte 20. The general formula is:

position of the nth record = (RN - 1) x RL

RN = record number
RL = record length

This calculation and positioning is, of course, only possible if all the records are of equal length. For this reason, all data (including numbers) are stored as character strings of exactly defined length; this is often limited to 255 bytes.

Note:

If, as is usual, the numbering of the records starts with 0, the following formula is obtained:

position of the nth record = RN x RL

Just like sequential files, random files must be opened before processing and closed afterwards. In addition, reading or writing a record must be done by giving the record number. An internal pointer is first positioned on the appropriate record, and then you can read or write.

The commands for processing random files are again, unfortunately, different for each version of BASIC. We are therefore going to look at each version separately, by means of a small example. Let's take up our example from Chapter 3 again. A phone directory consisting of names and numbers is to be realized as a random file. We'll choose the following record length quite arbitrarily:

name:	10 bytes
number:	8 bytes
length of record:	18 bytes

Figure 5.2 shows an example of a random file with two entries.

We are now going to create and process this file in the four BASIC versions, and call it RFILE (random file).

Microsoft

Microsoft BASIC is the best adapted to the processing of random files, as it offers the possibility of defining not only the whole record but also its components as items. The FIELD-command has the following general form:

```
FIELD file number, length of 1st field AS name of 1st field
                   length of 2nd field AS name of 2nd field
                   etc...
```

In our example we write:

```
FIELD#1,10 AS N$,8 AS P$
```

Thus a record of length 18 is defined, which is divided into two fields with the names N$ and P$. These fields are assigned values by the commands LSET or RSET. Figure 5.3 shows how the following commands store the values either left justified or right:

a)
```
LSET N$="MARY"
LSET P$="2325"
```

b)
```
RSET N$="MARY"
RSET P$="2325"
```

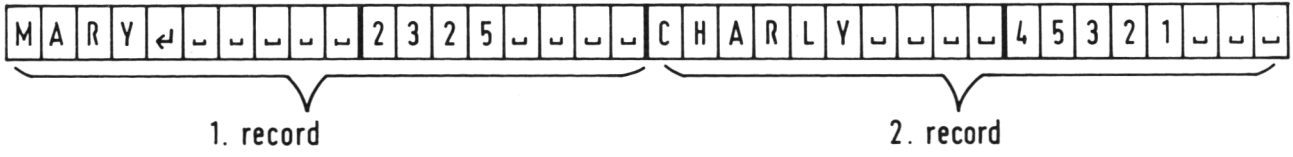


Fig.5.2; phone directory with two entries, organized as random file

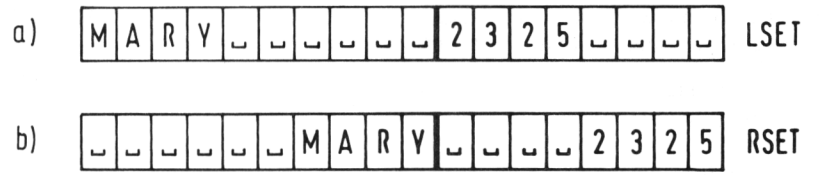


Fig.5.3; example of storing left or right justified

In future, we will use left justified storage only.

With the FIELD instruction an internal **buffer** is defined, whose content is written to the file with the PUT-command. This has the following general form:

PUT file number, record number

If, for example, you want to write the contents of the buffer to the file as the 5th record with the logical number 1, then you give the following command:

```
PUT#1,5
```

Reading is achieved with the GET-command, which has the same form:

GET file number, record number

The data read is subsequently to be found in the buffer defined by FIELD.

Finally, there are also the commands LOC and LOF, which are particularly useful with random files. LOC gives the number of the last record read or written, LOF the length of a file.

The following program creates the random file shown in Figure 5.2. The parameter LEN in the OPEN command gives the length of the file.

```
10 OPEN"RFILE"AS#1 LEN=18
20 FIELD#1,10 AS N$,8 AS P$
30 LSET N$="MARY":LSET P$="2325"
40 PUT#1,1
50 LSET N$="CHARLY":LSET P$="45321"
60 PUT#1,2
70 CLOSE 1
80 END
```

With the PUT command, you can, of course, give the record number by a variable RN, in which case you must make the following alterations:

```
35 LET RN=1
40 PUT#1,RN
55 LET RN=RN+1
60 PUT#1,RN
```

Note on Number Presentation and Conversion in Microsoft-BASIC

If a numerical value is to be stored in a random file, it must first be converted into a character string. There are three special functions for this purpose:

```
MKI$ (integer expression)----------> 2 bytes
MKS$ (single precision expression)---------- > 4 bytes
MKD$ (double precision expression)---------> 8 bytes
```

Example:

```
10 FIELD#1,4 AS X$,8 AS Y$
20 LSET X$=MKS$(X)
30 LSET Y$=MKD$(Y)
```

In X there should be a value with single precision, in Y a value with double precision. Both are converted into character strings.

The reconversion of character strings into numerical values is achieved with the following functions:

```
CVI (2-byte character string)
CVS (4-byte character string)
CVD (8-byte character string)
```

The following example shows the reconversion of the character strings from the above example:

Example:

```
10 FIELD#1,4 AS X$,8 AS Y$
20 GET#1,RN:REM RN=RECORD NUMBER
30 X=CVS(X$)
40 Y=CVD(Y$)
```

Apple

With Apple, you give a length-of-record parameter when opening a file. When reading or writing a record, a record number parameter R is required.

You yourself must divide the record into its single data fields. If the relevant character

string is shorter than the length provided, then this must be filled with blanks. This ensures that the phone number always begins in the 11th position. We will agree on a length of 19 characters, as the RETURN symbol is also stored. In lines 60 and 100, the values of N$ and P$ are concatenated.

```
 10 LET D$=CHR$(4)
 20 PRINT D$;"OPEN";"RFILE,L19"
 30 LET N$="MARY                    "
 40 LET RN=1
 50 PRINT D$;"WRITE";"RFILE,R";RN
 60 LET A$=N$+P$
 70 PRINT A$
 80 LET N$="CHARLY      ":LET P$="45321     "
 90 LET RN=RN+1
100 LET A$=N$+P$
110 PRINT D$;"WRITE";"RFILE,R";RN
120 PRINT A$
130 PRINT D$;"CLOSE";"RFILE"
140 END
```

Commodore

Just as with BBC, Commodore provides no support for random files in its system, so you will have to do rather more programming, and gain access to the desired record with **positioning commands**.

Opening a random file is done with a modified OPEN command. Using the CHR$ function, you define the length of the record, **which must be one more than the actual length**, as the RETURN symbol is stored, too. In our example you would write:

```
OPEN 1,8,2,"RFILE,L,"+CHR$(19)
```

A random file must be **created** before it can be used for the first time. If, for example, you want to store a maximum of 50 records, then you position on the 50th record, and write the character CHR$(255) into it. The positioning command is given through the command channel 15, and has the following general form:

```
PRINT#file no.,"P"+CHR$(channel no.)+CHR$(low)+CHR$(high)+CHR$(byte no.)
```

The file number is the actual number you gave with the OPEN command for opening command channel 15. The channel number is the number given when opening the file.

The record number is divided into two bytes "high" and "low". If the record number is smaller than 256 (which will always be the case in our example programs), then the high-byte is given the value 0 and the low-byte the record number as value. Greater record numbers must be resolved to multiples of 256.

Example:

Record number = 300
Resolution: 300 = 1 x 256 + 44
High-byte: 1
Low-byte: 44

The byte number, finally, serves to position on a particular byte in the record. Normally the number 1 is given, as numbering begins with 1 and not 0 in the case of Commodore. Reading and writing individual records is also done with the help of the positioning command.

```
 10 OPEN 1,8,2,"RFILE,L,"+CHR$(19)
 20 OPEN 3,8,15
 30 PRINT#3"P"+CHR$(2)+CHR$(50)+CHR$(0)+CHR$(1)
 40 PRINT#1,CHR$(255):REM RESERVE 50 RECORDS
 50 LET N$="MARY          ":LET P$="2325      "
 60 LET A$=N$+P$
 70 LET RN=1
 80 PRINT#3,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
 90 PRINT#1,A$
100 LET N$="CHARLY        ":LET P$="45321     "
110 LET A$=N$+P$
120 LET RN=RN+1
130 PRINT#3,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
140 PRINT#1,A$
150 CLOSE 1:CLOSE 3
160 END
```

Please note that in line 60 the two character strings in N$ and P$ must be set together with no space between them. Further, the command channel 15, which has been given the logical number 3, must be closed again in line 150.

Two further points must be noted with Commodore: firstly, only records with a maximum of 88 bytes can be read with the INPUT# command; secondly, no more than one random file may be opened (although a sequential file can be opened in addition).

BBC

With BBC BASIC, you must add two to the length of the record, as type and length are stored in addition to each record. Figure 5.4 shows an example. The byte T gives the type of data, the byte L the length of the record. (In fact, records are stored backwards on the file, but this makes no difference to the programming.)

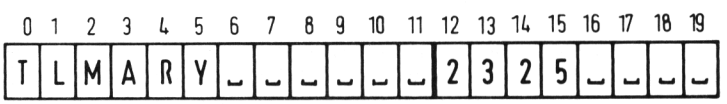


Fig.5.4; storing a record with BBC

As with Commodore, you must give a position command before reading or writing, which is done here with the command PTR#X. You do **not**, however, give the record number, but the **byte position** relative to the beginning of the file. With the PRINT# command you then write the record to the file.

```
 10 X=OPENOUT"RFILE"
 20 LET N$="MARY          ":LET P$="2325      "
 30 LET A$=N$+P$
 40 PTR#X=20
 50 PRINT#X,A$
 60 LET N$="CHARLY        ":LET P$="45321     "
 70 LET A$=N$+P$
 80 PTR#X=40
 90 PRINT#X,A$
100 CLOSE#X
110 END
```

In BBC BASIC the numbering of the records begins with 0. In order to remain compatible with those systems beginning with 1, however, we will forgo the first record.

A further useful command in connection with random files is called EXT#X (= extent or length). It gives the length of a file in bytes. For example, it can be used (with sequential files, too) to append records to the end of the file, by writing:

```
PTR#X=EXT#X
```

If you open an already existing file with the command OPENOUT, the contents will be deleted. To avoid this, use the command OPENUP, which opens a file for reading or writing. This command, however, is only implemented in the version BASIC II.

Please note that, when a file is newly created, 64 sectors are reserved on the diskette. Each sector contains 256 bytes. If you know the length of the records, you can calculate from these facts the maximum number of records. As the internal data buffer is 256 bytes long, it is convenient to choose a multiple or submultiple of 256 for the length of the record, e.g., 512, 256,64,32 or 16 bytes. In this way you make the best use of the room on the diskette.

5.2 Example: A Stock Inventory

As an example, let's look at the administration of a small stockroom, in which a maximum of 50 articles (e.g., electronic parts) is kept in stock. Each article is marked by four characteristics:

Number of Article:	1 - 50 (maximum of two digits)
Name of Article:	maximum of 25 characters

Price per Piece:	maximum 999.99 (= 6 characters)
Amount:	maximum 9999 (= 4 characters)

The article numbers (or part numbers) should be identical with the record numbers. This has the advantage that the article number **need not be stored** and that the file is sorted in ascending order in regard to the article numbers. Thus each record consists of 25+6+4=35 characters. We will add one byte to the beginning of each record, informing us whether the record is empty or occupied. We will choose the following identification:

1. byte = "E" = record is empty.

1 .byte = "%" = record is occupied.

Figure 5.5 shows an example of the contents of a record. You find the names of the individual variables. The complete record is stored in the variable R$, and the separate fields are indicated by C$,D$,V$ and Q$. The length of the record is 36 bytes.

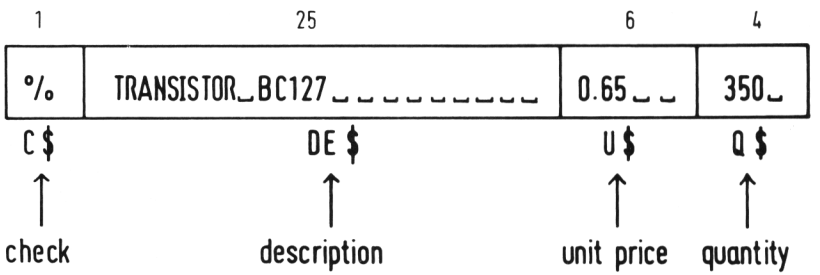


Fig. 5.5; example of a record

We'll call the file IFILE (inventory file). The program is to perform the following tasks:

1. Initialization of the file IFILE
2. Creation of a new record
3. Printing of contents of IFILE
4. Direct access to an article and alteration of amount given, if desired.

You can see the structure diagram in Figure 5.6. In contrast to our previous programs, the file **is only opened once in the main program.** The individual subroutines, according to the task, access the **file for reading or writing.** The file is only closed with the last subroutine "end". Furthermore you will see that the subroutines are called upon several times for reading, writing and display.

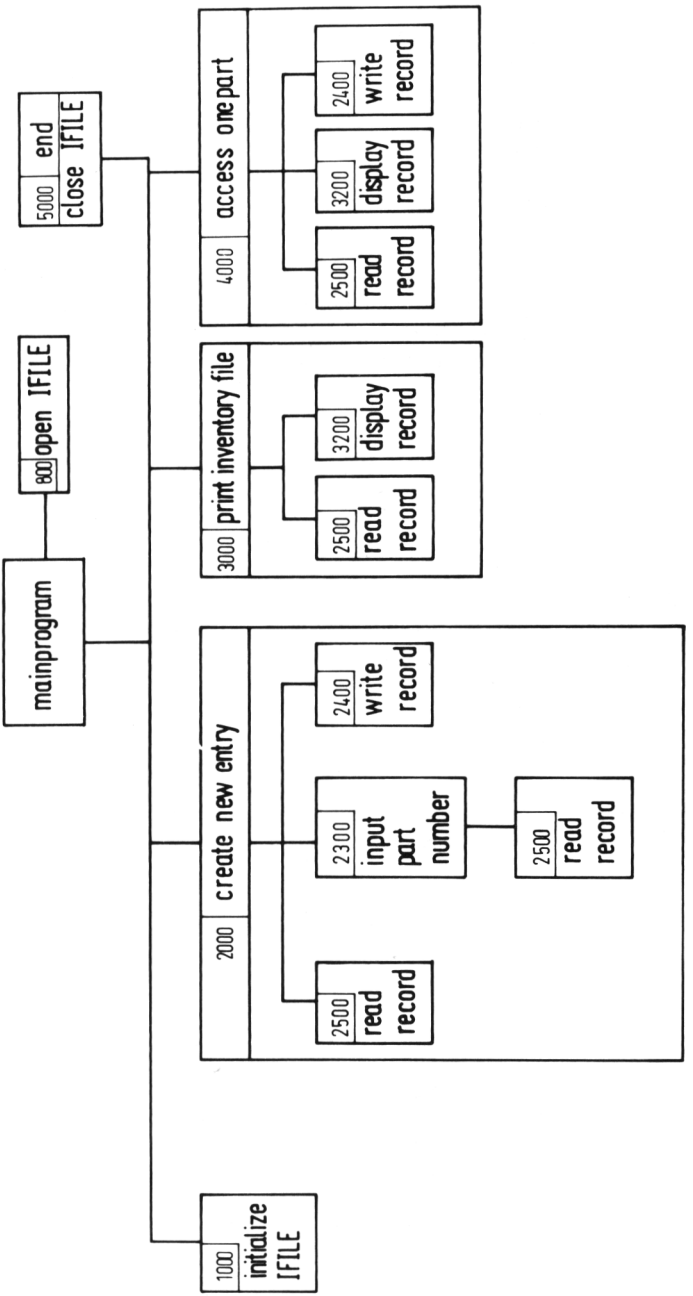


Fig.5.6; structure diagram for inventory

Let's look at the main program. In line 30 the variable BP$ is assigned the character with the code number 7, which is interpreted by many computers as a **beep tone.** We're going to cause this tone every time a user makes an invalid input (e.g., when the number 6 is entered with the menu). If your computer has other commands for producing the beep tone, you can, of course, also use these (e.g. SOUND). The variables BD$,BU$ and BQ$ are assigned exactly as many blanks as the fields DE$,U$ and Q$ are long (see Figure 5.5). We need them in case the character strings given in the dialog have to be filled up with blanks.

The menu is programmed by the method you are already familiar with. Only point 5 differs a little: first the file IFILE is closed in line 5000; then follows the END command (see lines 5000 - 5120). In line 810 the value 37 is given as length in opening the file IFILE, as the RETURN symbol is also stored.

Note for Microsoft BASIC

If you are working with this BASIC system, you can save a lot of commands and program more simply. In line 811 write:

```
811 FIELD#1,1 AS C1$, 25 AS D1$, 6 AS U1$, 4 AS Q1$
```

The variables mentioned here are given their values by LSET commands. Filling the character strings with blanks becomes unnecessary. In this way, the programming of points 2, 3 and 4 in particular is simplified. In order to guarantee a rather more uniform presentation, we have not used these advantages in the programs. However, you will find alternative suggestions at the end of each appropriate program.

Apple:

```
 10 REM=====INVENTORY=====
 20 LET F$="IFILE"
 21 LET D$=CHR$(4)
 30 LET BP$=CHR$(7):REM BEEP
 40 LET BD$="                          ":REM 25 BLANKS
 50 LET BU$="                    ":REM 6 BLANKS
 60 LET BQ$="              ":REM 4 BLANKS
 70 GOSUB 800:REM OPEN IFILE
 80 GOSUB 500
 90 PRINT TAB(10);"INVENTORY"
100 PRINT:PRINT
110 PRINT TAB(10);"1=INITIALIZE INVENTORY FILE"
120 PRINT TAB(10);"2=CREATE NEW ENTRY"
130 PRINT TAB(10);"3=PRINT INVENTORY"
140 PRINT TAB(10);"4=ACCESS ONE PART"
150 PRINT TAB(10);"5=END"
160 PRINT:PRINT
170 PRINT TAB(10);"SELECT NUMBER, PLEASE"
180 INPUT N
190 IF N<1 OR N>5 THEN PRINT BP$:GOTO 180
200 ON N GOSUB 1000,2000,3000,4000,5000
210 GOTO 80
```

```
                500 REM=====CLEAR SCREEN=====
Apple:          510 HOME
MS,BBC:         510 CLS
CBM:            510 PRINT CHR$(147)
                520 RETURN
```

```
                800 REM=====OPEN IFILE=====
Apple:          810 PRINT D$;"OPEN";F$;",L37"
MS:             810 OPEN F$ AS#1 LEN=36
                811 FIELD#1,36 AS RC$
CBM:            810 OPEN 1,8,2,F$+",L,"+CHR$(37)
                811 OPEN 3,8,15
                812 PRINT#3,"P"+CHR$(2)+CHR$(51)+CHR$(0)+CHR
                    $(1)
                813 PRINT#1,CHR$(255)
BBC:            810 X=OPENUP F$
                820 RETURN
```

5.3 Initializing the File IFILE

First of all, let's initialize the file IFILE by writing the letter E (for "empty") to the first byte of each record. This has two reasons: firstly, with some computers (e.g. Apple) you can only read a record from a random file **if it has been written already**. If this is not the case, you will get a negative message such as END OF DATA. Secondly, our program uses this indicator with the letter E to differentiate between empty and written records.

```
               1000 REM=====INITIALIZE IFILE=====
               1010 GOSUB 500
               1020 PRINT"INVENTORY FILE WILL BE DELETED"
               1030 INPUT"TYPE Y TO CONFIRM";A$
               1040 IF A$<>"Y"THEN RETURN
BBC:           1045 CLOSE#X:X=OPENOUT F$
               1050 LET C$="E":REM EMPTY
               1060 FOR I=1 TO 35
               1070 LET C$=C$+" ":REM ADD 35 BLANKS
               1080 NEXT I
               1090 FOR I=1 TO 50
Apple:         1100 PRINT D$;"WRITE";F$;",R";I
               1101 PRINT C$
MS:            1100 LSET RC$=C$
               1101 PUT#1,I
CBM:           1100 PRINT#3,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(1)
               1101 PRINT#1,C$
BBC:           1100 PTR#X=38*I
               1101 PRINT#X,C$
               1110 NEXT I
Apple:         1111 PRINT D$
               1120 RETURN
```

5.4 Create new Entry

In line 2020 we call the subroutine for reading a record beginning at line 2300. First the part number PN is read in and checked. The subroutine from line 2500 onwards reads the appropriate record.

Now we check in line 2030 whether the record is empty. If this is not the case, the user is informed and can then decide whether to overwrite the record or not. In this way you can delete an old article from the file and put a new one in its place with the same article number.

In line 2070, the record is marked with the percent symbol as being occupied. Then in the dialog we read in name, price per piece and amount. A check is made each time to ensure that the character string entered does not exceed the prescribed length.

Filling up with blanks, as in lines 2100,2130 and 2160, is rather more complicated to understand. Let's look at line 2160 as a detailed example. Suppose the number 25 is given as the amount; then the value of the variable Q$ consists of two characters and must be filled up with two blanks. The function MID$ selects the right quantity from the string of blanks stored in BQ$:

```
MID$(BQ$,1,4-2)
```

Two blanks are selected and appended to the character string in R$. Now you can see why we stored the appropriate strings of blanks in the variables BD$,BU$ and BQ$ at the beginning of the program.

Now that the construction of the character string in R$ has been completed, it is written to IFILE in the subroutine 2400-2420 in the position designated by part number PN.

```
2000 REM =====CREATE NEW ENTRY====
2010 GOSUB 500
2020 GOSUB 2300:REM READ RECORD FROM IFILE TO R$
2030 IF LEFT$(R$,1)="E" THEN 2070
2040 PRINT"RECORD NOT EMPTY"
2050 INPUT"OVERWRITE(Y/N)";A$
2060 IF A$<>"Y"THEN RETURN
2070 LET R$="%":REM RECORD NOT EMPTY
2080 INPUT"DESCRIPTION";DE$
2090 IF LEN(DE$)>25 THEN PRINT BP$:GOTO 2080
2100 LET R$=R$+DE$+MID$(BD$,1,25-LEN(DE$))
2110 INPUT"UNIT PRICE";U$
2120 IF LEN(U$)>6 THEN PRINT BP$:GOTO 2110
2130 LET R$=R$+U$+MID$(BU$,1,6-LEN(U$))
2140 INPUT"QUANTITY";Q$
2150 IF LEN(Q$)>4 THEN PRINT BP$:GOTO 2140
2160 LET R$=R$+Q$+MID$(BQ$,1,4-LEN(Q$))
2170 INPUT"CORRECT(Y/N)";A$
2180 IF A$<>"Y"THEN 2070
```

```
2190 GOSUB 2400:REM WRITE RECORD FROM R$ TO IFILE
2200 RETURN

2300 REM=====INPUT PART# AND READ RECORD==
     ===
2310 INPUT"PART NUMBER";PN
2320 IF PN <1 OR PN >50 THEN PRINT BP$:GOTO 2310
2330 GOSUB 2500:REM READ RECORD
2340 RETURN

2400 REM=====WRITE RECORD FROM R$ TO IFILE==
     ===
```

Apple:	2410 PRINT D$;"WRITE";F$;",R";PN 2411 PRINT R$ 2412 PRINT D$
MS:	2410 LSET RC$=R$ 2411 PUT#1,PN
CBM:	2410 PRINT#3,"P"+CHR$(2)+CHR$(PN)+CHR$(0)+CHR$(1) 2411 PRINT#1,R$
BBC:	2410 PTR#X=38*PN 2411 PRINT#X,R$

```
2420 RETURN

2500 REM=====READ RECORD FROM IFILE TO R$==
     ===
```

Apple:	2510 PRINT D$;"READ";F$;",R";PN 2511 INPUT R$ 2512 PRINT D$
MS:	2510 GET#1,PN 2511 LET R$=RC$
CBM:	2510 PRINT#3,"P"+CHR$(2)+CHR$(PN)+CHR$(0)+CHR$(1) 2511 INPUT#1,R$
BBC:	2510 PTR#X=38*PN 2511 INPUT#X,R$

```
2520 RETURN
```

Microsoft Alternative

The laborious construction of the character string in R$ is dispensed with (lines 2070,2100,2130,2160). Instead, you write:

```
2070 LSET C$="%"
2100 LSET D1$=DE$
```

```
2130 LSET U1$=U$
2160 LSET Q1$=Q$
```

You alter the subroutine for writing as follows:

```
2410 is left out.
```

It would be even better to read in, for example, the amount figure as a number in a variable Q, and convert this into a character string with the command MKI$:

```
2160 LSET Q1$=MKI$(Q)
```

The same goes for the other values. Please note, however, that an integer expression is converted into a 2 byte character string, while a decimal fraction is converted into 4 bytes with the function MKS$.

5.5 Printing the Inventory File

We read the 50 records sequentially and print the ones which aren't empty (line 3060). The subroutine 3200-3270 "pulls apart" the character string from R$ into its three components. Figure 5.7 shows an example.

```
3000 REM=====PRINT INVENTORY=====
3010 GOSUB 500
3020 PRINT"INVENTORY FILE"
3030 PRINT"==============="
3040 FOR PN=1 TO 50
3050 GOSUB 2500:REM READ RECORD FROM IFILE TO R$
3060 IF LEFT$(R$,1)="E" THEN 3080
3070 GOSUB 3200:REM DISPLAY RECORD ON SCREEN
3080 NEXT PN
3090 INPUT"CONTINUE(Y/N)";A$
3100 IF A$<>"Y"THEN 3090
3110 RETURN

3200 REM=====DISPLAY RECORD ON SCREEN===
     ==
3210 LET DE$=MID$(R$,2,25)
3220 LET U$=MID$(R$,27,6)
3230 LET Q$=MID$(R$,33,4)
3240 PRINT"PART#";PN
3250 PRINT DE$:PRINT U$:PRINT Q$
3260 PRINT:PRINT
3270 RETURN
```

Microsoft Alternative

Lines 3210-3230 are left out. In line 3250 write:

```
3250 PRINT D1$:PRINT U1$:PRINT Q1$
```

If you have used the functions MKI$ and MKS$ in Point 2, you must now write:

```
3250 PRINT D1$:PRINT CVS(U1$):PRINT CVI(Q1$)
```

```
 INVENTORY FILE
==============
PART#1
TRANSISTOR BC 127
0.12
250


PART#15
LED-DISPLAY
1.25
80


PART#25
TRANSISTOR BC 107
0.30
522


PART#45
CRT-CONTROL
2.10
25
```

Fig.5.7; example of a print-out of an inventory file

5.6 Gaining Access to an Article

The following program (lines 4000 to 4240) performs three tasks:

1. It selects directly one desired record and displays its contents (if it isn't empty).

2. You can increase the amount of the article read (add to stock).

3. You can reduce the amount (subtract from stock).

In line 4020 the subroutine for reading a record is called up. If the record read is empty, this is announced, and after a short pause we branch off to the main menu. If it isn't empty, the contents of the record are displayed on the screen (subroutine 3200-3270).

The user now enters either A or S or R. The letter R causes a return to the main menu (line 4140). If the letter S is chosen, a negative number must be entered (line 4150).

The character string in Q$ is converted in line 4170 into a numerical value with the command VAL. The number entered, N, is added; the result is reconverted into a character string after a check (line 4190) and appended to the first 32 bytes of R$ (line 4210). The subroutine 2400-2420 writes the altered record back.

```
4000 REM=====ACCESS ONE PART=====
4010 GOSUB 500
4020 GOSUB 2300:REM READ RECORD
4030 IF LEFT$(R$,1)="%"THEN 4080
4040 PRINT"RECORD EMPTY":PRINT BP$
4050 FOR I=1 TO 1000
4060 NEXT I:REM WAIT
4070 RETURN
4080 GOSUB 3200:REM DISPLAY RECORD ON SCREEN
4090 PRINT"A=ADD TO STOCK"
4100 PRINT"S=SUBTRACT FROM STOCK"
4110 PRINT"R=RETURN TO MENU"
4120 INPUT"YOUR CHOICE";A$
4130 IF A$ <>"A"AND A$ <>"S"AND A$<>"R"THEN 4120
4140 IF A$="R"THEN RETURN
4150 IF A$="S"THEN PRINT"TYPE NEGATIVE NUMBER"
4160 INPUT"QUANTITY";N
4170 LET Q=VAL(Q$):REM CONVERT TO INTEGER
4180 LET Q=Q+N
4190 IF Q<0 OR Q>9999 THEN PRINT BP$:GOTO 4160
4200 LET Q$=STR$(Q):REM CONVERT TO STRING
MS,CBM:
4201 LET Q$=RIGHT$ (Q$,LEN(Q$)-1)
4210 LET S$=LEFT$(R$,32)+Q$+MID$(BQ$,1,4-LEN(Q$))
4220 LET R$=S$
4230 GOSUB 2400:REM WRITE RECORD TO IFILE
4240 RETURN
```

```
                    5000 REM=====CLOSE AND END=====
Apple:              5010 PRINT D$;"CLOSE";F$
MS,CBM:             5010 CLOSE 1
CBM:                5011 CLOSE 3
BBC:                5010 CLOSE#X
                    5020 END
```

Microsoft Alternative

If you have used the functions MKI$ and MKS$ to store amount and price per piece at Point 2, you must now convert the amount given into a numerical value with CVI.

```
4170 LET Q=CVI(Q1$)
```

After the addition comes the reconversion to a character string:

```
4200 LSET Q1$=MKI$(Q)
```

Lines 4210 and 4220 are left out.

Note

In some BASIC versions (e.g. MS, CBM and others), the function STR$ produces a leading blank. The function RIGHT$ in line 4201 is used to remove it. We'll use this technique throughout chapters 6 to 8.

Chapter 6

Index Sequential Files

6.1 Example: A Book Catalogue

Let's just recap shortly. First you met with the unsorted sequential files. The insertion of new records is very simple, since this is done at the end of the file. The disadvantage, however, is that printing sorted long sequential files is arduous and time-consuming. Sorted sequential files avoid this disadvantage, but on the other hand, inserting records in the right place is a problem, as we saw in Section 4.4.

The index sequential file avoids both disadvantages. New records can be easily inserted, and the sorted printing is also relatively simple. There is a further advantage. A sorted sequential file can, of course, only be sorted in respect to one kind of order, e.g., alphabetically in respect to name. With the index sequential method it is possible to sort according to several kinds of order, called **keys**.

Using a book catalog as an example, we'll look at the index sequential method. Let us suppose you keep your books at home in several shelves, labelled A,B,C etc. A maximum of 99 books is to go on each shelf, so that the place where each book is to be found can be given by a letter and a one or two-digit number.

Examples:

Place number: B25 The book is in the 25th position on shelf B
Place number: F9 The book is in the 9th position on shelf F

Besides the place number, we're going to store the book's title and the name of the author. We choose the following maximum lengths:

Title:	maximum of 30 characters
Author:	maximum of 20 characters
Place number:	maximum of 3 characters
Total:	53 characters

Exactly as in the last chapter, we are going to mark an empty record with the letter E and an occupied one with the percent symbol %. This check byte is placed before the

title, so that the record is now 54 bytes long. Figure 6.1 shows an example. We'll call the file BFILE (book file). As the RETURN symbol will also be stored, we will give the length 55 in the program.

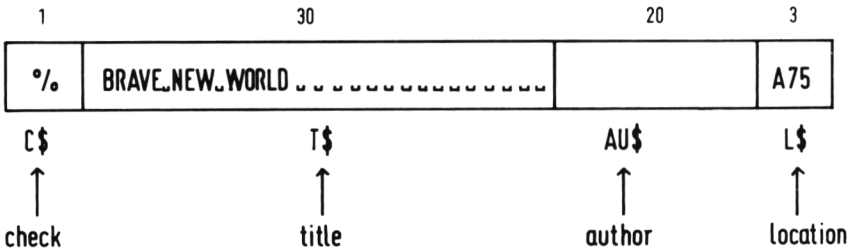


Fig. 6.1; Example of a record in the file BFILE.

We construct our file as a random file, and call it the **main file**. In addition we will create a so-called **index file**, and call it IXFILE. In practice, the records of the main file consist of many fields. We could, for example, store the name of the publisher, the year of publication and the ISB-number along with each book. The index file contains much less information. It is created as a **sequential file** in which the records consist of exactly **two** fields: the **key** and the **record number**. Let us choose the name of the author as our key. Figure 6.2 shows an example. On the left you can see the main file, and to the right of it the index file.

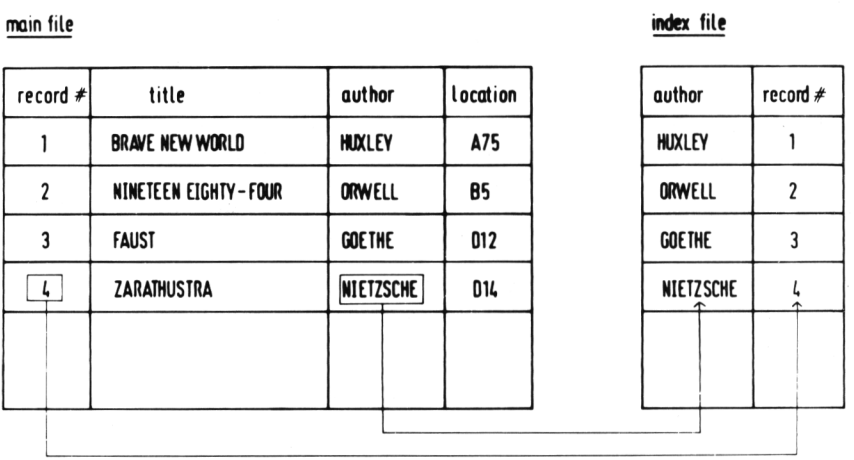


Fig. 6.2; example of a main file and index file

Every time an entry is made in the main file, it is also made in the index file. With large computers, this often happens automatically (e.g. ISAM files; ISAM= Index Sequential Access Method), but with microcomputers you must handle the index file yourself.

You have free choice of access to the records in an index sequential file, but this must be done in two steps:

1. Sequential access to the index file.
2. Direct access to the main file.

Example:

You are looking for George Orwell's book "1984". Search for the name "Orwell" in the index file. You read this record by record, i.e. sequentially, and find the appropriate record number 4. Now go directly to the 4th record of the main file and read all the information on this book into the workspace.

The essential advantage of the index sequential method, therefore, consists in searching for a given key in a relatively small **index file** and finding the appropriate number there. If the index file is not too big, it can be kept **permanently in the workspace**, enabling you to search or sort very quickly.

If you would also like to use the title as a key, simply create a second index file. Finally, you could create a third file for the book position. In this case you would have a **completely inverted main file**, and you can search and sort according to each of the three criteria "Author","Title" and "Position on shelf".

Admittedly, this method has a small disadvantage which ought not to be disregarded. With every alteration to the main file, you must update **all** the index files. As this can, in certain circumstances, be very time-consuming, we usually keep to few index files in practice. In the following program, we shall be working with only one index file.

We want a program that will perform the following tasks:

1. Initializing the main file BFILE
2. Entering new records
3. Printing out unsorted the contents of BFILE
4. Printing out the contents of BFILE sorted alphabetically according to author.
5. Printing out the books of the author required

As we are only going to have a look at the principles involved in processing index sequential files, let's limit ourselves to a file with a maximum of 99 records. You can increase the number as you like, until you arrive at the limits of the storage capacity of your diskette.

We'll now look at the main program. It is very similar to the program in the last chapter, so you can use this as a basis, and only have to alter a few commands. In lines 40-60,

character strings are again defined, consisting of blanks and serving as fillers. The two arrays defined in line 70 (N$ and RN) are used to store the index file.

In contrast to the program in the last chapter, the file BFILE is not opened at the beginning of the main program and only closed after processing, but is opened and closed by each subroutine individually. The reason for this is that we will need a second file IXFILE, and that with some computers it is not possible to have more than one file opened at the same time. If your computer permits the opening of several files, you can simplify the program accordingly. Just as in the last example, you open the file BFILE in the main program and do not close it again until line 6000. Opening and closing in the subroutines is dispensed with.

```
                10 REM=====BOOK FILE=====
                20 LET F$="BFILE":LET I$="IXFILE"
Apple:          21 LET D$=CHR$(4)
                30 LET BP$=CHR$(7):REM BEEP
                40 LET BT$="                              ":REM 30 BLANKS
                50 LET BA$="                    ":REM 20 BLANKS
                60 LET BL$="   ":REM 3 BLANKS
                70 DIM N$(99),RN(99)
                80 GOSUB 500
                90 PRINT TAB(10);"BOOK FILE"
               100 PRINT:PRINT
               110 PRINT TAB(10);"1=INITIALIZE BOOK FILE"
               120 PRINT TAB(10);"2=CREATE NEW ENTRY"
               130 PRINT TAB(10);"3=PRINT UNSORTED"
               140 PRINT TAB(10);"4=PRINT SORTED"
               150 PRINT TAB(10);"5=SEARCH FOR AUTHOR"
               160 PRINT TAB(10);"6=END"
               170 PRINT:PRINT
               180 PRINT TAB(10);"SELECT NUMBER,PLEASE"
               190 INPUT N
               200 IF N<1 OR N>6 THEN PRINT BP$: GOTO 190
               210 ON N GOSUB 1000,2000,3000,4000,5000,6000
               220 GOTO 80

               500 REM=====CLEAR SCREEN=====
Apple:         510 HOME
MS,BBC:        510 CLS
CBM:           510 PRINT CHR$(147)
               520 RETURN

               800 REM=====OPEN BFILE=====
Apple:         810 PRINT D$;"OPEN";F$;",L55"
MS:            810 OPEN F$ AS#1 LEN=54
               811 FIELD#1,54 AS RC$
CBM:           810 OPEN 1,8,2,F$+",L,"+CHR$(55)
               811 OPEN 3,8,15
```

```
CBM:    812 PRINT#3,"P"+CHR$(2)+CHR$(100)+CHR$(0)+CHR$(1)
        813 PRINT#1,CHR$(255)
BBC:    810 X=OPENUP F$
        820 RETURN

        900 REM=====CLOSE BFILE=====
Apple:  910 PRINT D$;"CLOSE";F$
MS,CBM: 910 CLOSE 1
CBM:    911 CLOSE 3
BBC:    910 CLOSE#X
        920 RETURN
```

Figures 6.3, 6.4 and 6.5 show the structure diagram laid out in several pieces. You can see that the program is very highly structured and consists of numerous subroutines. The subroutines 800 and 900 for opening and closing the file BFILE are needed in each part of the menu. Routines 2300, 2400, 2500 and 3200 you already know from the last chapter; they must be altered only minimally. New additions are the routines 2600, 4400 and 2700, which open or close the index file, and also routine 4600, which sorts the array N$ using the bubble sort technique.

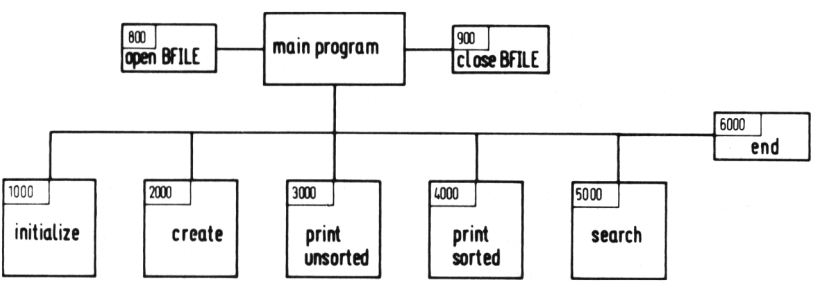


Fig.6.3; structure diagram for the complete program.

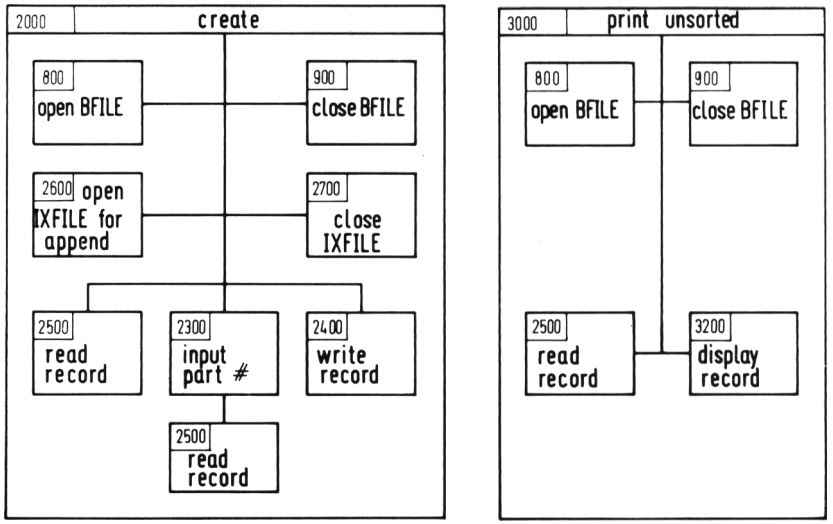


Fig.6.4; structure diagram for the subroutines CREATE and PRINT UNSORTED.

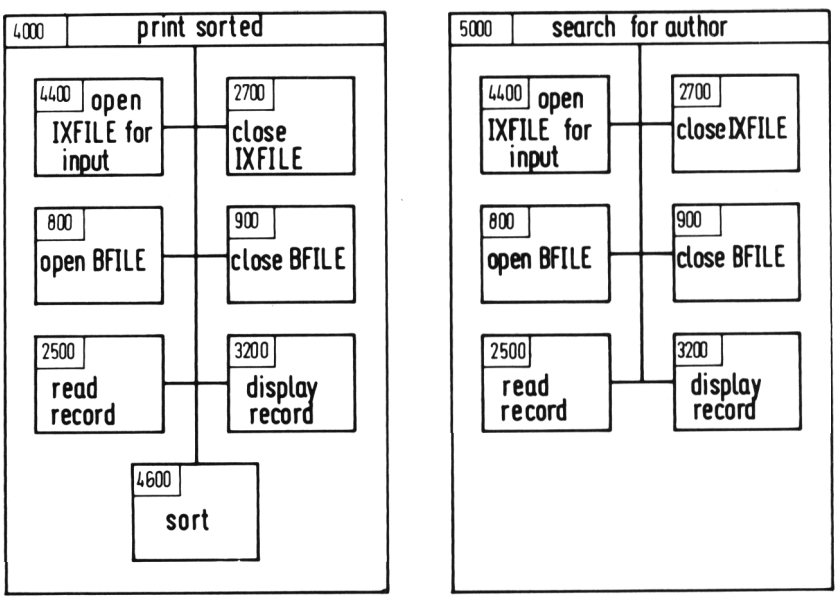


Fig.6.5; structure diagram for the subroutines PRINT SORTED and SEARCH.

6.2 Initializing the File BFILE

The program is, in essentials, identical to that in the previous chapter. You must undertake the following alterations:

1. In line 1060, 53 blanks are appended.
2. In line 1090, 99 records are initialized.
3. From line 1130, the index file is opened for writing and then closed again at This provides the opportunity of adding more records later with

```
1000 REM=====INITIALIZE BFILE=====
1010 GOSUB 500
1020 PRINT "BOOK FILE WILL BE DELETED"
1030 INPUT "TYPE Y TO CONFIRM";A$
1040 IF A$<>"Y" THEN RETURN
1045 GOSUB 800:REM OPEN BFILE
BBC: 1045 X=OPENOUT F$
1050 LET C$="E":REM EMPTY
1060 FOR I=1 TO 53
1070 LET C$=C$+" ":REM ADD 53 BLANKS
1080 NEXT I
1090 FOR I=1 TO 99
Apple: 1000 PRINT D$;"WRITE";F$;",R";I
       1101 PRINT C$
MS:    1100 LSET RC$=C$
       1101 PUT#1,I
CBM:   1100 PRINT#3,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(1)
       1101 PRINT#1,C$
BBC:   1100 PTR#X=56*I
       1101 PRINT#X,C$
1110 NEXT I
1120 GOSUB 900:REM CLOSE BFILE
Apple: 1130 PRINT D$;"OPEN";I$
       1131 PRINT D$;"WRITE";I$
       1132 PRINT D$;"CLOSE";I$
MS:    1130 OPEN I$ FOR OUTPUT AS#2
       1131 CLOSE 2
CBM:   1130 OPEN 2,8,3,I$+",S,W"
       1131 CLOSE 2
BBC:   1130 Y=OPENOUT I$
       1131 CLOSE#Y
1140 RETURN
```

6.3 Entering a New Book

This program, too, is essentially identical to that in Chapter 5. After the record has been written to the file BFILE, however, the author's name and the record number must be entered into the index file. You do this in lines 2200 to 2220. Note also that there are two new subroutines 2600 and 2700, which open and close the index file respectively.

```
2000 REM=====CREATE NEW ENTRY=====
2005 GOSUB 800:REM OPEN BFILE
2010 GOSUB 500
2020 GOSUB 2300:REM READ RECORD FROM BFILE TO R$
2030 IF LEFT$(R$,1)="E" THEN 2070
2040 PRINT"RECORD NOT EMPTY"
2050 INPUT"OVERWRITE(Y/N)";A$
```

Apple:	2060 IF A$<>"Y" THEN RETURN
MS:	2060 IF A$<>"Y" THEN CLOSE:RETURN
CBM:	2060 IF A$<>"Y" THEN GOSUB 900:RETURN
BBC:	2060 IF A$<>"Y" THEN CLOSE#X:RETURN

```
2065 GOSUB 2800:REM REMOVE RECORD FROM IXFILE
2070 LET R$="%":REM RECORD NOT EMPTY
2080 INPUT"TITLE";T$
2090 IF LEN(T$)>30 THEN PRINT BP$:GOTO 2080
2100 LET R$=R$+T$+MID$(BT$,1,30-LEN(T$))
2110 INPUT"AUTHOR";AU$
2120 IF LEN(AU$)>20 THEN PRINT BP$:GOTO 2110
2130 LET R$=R$+AU$+MID$(BA$,1,20-LEN(AU$))
2140 INPUT"LOCATION";L$
2150 IF LEN(L$)>3 THEN PRINT BP$:GOTO 2140
2160 LET R$=R$+L$+MID$(BL$,1,3-LEN(L$))
2170 INPUT"CORRECT(Y/N)";A$
2180 IF A$<>"Y" THEN 2070
2190 GOSUB 2400:REM WRITE RECORD FROM R$ TO
     BFILE
2200 GOSUB 900: REM CLOSE BFILE
2210 GOSUB 2600:REM OPEN IXFILE FOR APPEND
```

Apple:	2220 PRINT D$;"APPEND";I$ 2221 PRINT D$;"WRITE";I$ 2222 PRINT AU$:PRINT PN
MS,CBM:	2220 PRINT#2,AU$:PRINT#2,PN
BBC:	2220 PRINT#Y,AU$:PRINT#Y,PN

```
2230 GOSUB 2700:REM CLOSE XFILE
```

Apple:	2240 GOTO 80
Others:	2240 RETURN

```
2300 REM=====INPUT BOOK# AND READ RECORD=
     ====
2310 INPUT"BOOK NUMBER";PN
2320 IF PN<1 OR PN>99 THEN PRINT BP$:GOTO 2310
2330 GOSUB 2500:REM READ RECORD
2340 RETURN


2400 REM=====WRITE RECORD FROM R$ TO BFILE==
     ===
```

Apple:	2410 PRINT D$;"WRITE";F$;",R";PN 2411 PRINT R$ 2412 PRINT D$
MS:	2410 LSET RC$=R$ 2411 PUT#1,PN
CBM:	2410 PRINT#3,"P"+CHR$(2)+CHR$(PN)+CHR$(0)+CHR$(1) 2411 PRINT#1,R$
BBC:	2410 PTR#X=56*PN 2411 PRINT#X,R$

```
2420 RETURN
2500 REM =====READ RECORD FROM BFILE TO R$
     =====
```

Apple:	2510 PRINT D$; "READ";F$;",R";PN 2511 INPUT R$ 2512 PRINT D$
MS:	2510 GET#1,PN 2511 LET R$=RC$
CBM:	2510 PRINT#3,"P"+CHR$92)+CHR$(PN)+CHR$(0)+CHR$(1) 2511 INPUT#1,R$
BBC:	2510 PTR#X=56*PN 2511 INPUT#X,R$

```
2520 RETURN

2600 REM=====OPEN IXFILE FOR APPEND====
```

Apple:	2610 PRINT D$;"OPEN";I$
MS:	2610 OPEN I$ FOR APPEND AS#2
CBM:	2610 OPEN 2,8,3,I$+",S,A"
BBC:	2610 Y=OPENUP I$:PTR#Y=Y=EXT#Y

```
2620 RETURN

2700 REM=====CLOSE IXFILE====
```

Apple:	2710 PRINT D$; "CLOSE";I$
MS,CBM:	2710 CLOSE 2
BBC:	2710 CLOSE#Y

```
2720 RETURN
2800 REM=====REMOVE RECORD FROM IXFILE===
     =
```

```
                2810 GOSUB 2950:REM OPEN TEMP FOR OUTPUT
                2820 GOSUB 4400:REM OPEN IXFILE FOR INPUT
Apple:          2830 PRINT D$;"READ";I$
                2831 ONERR GOTO 2879
MS:             2830 IF EOF(2) THEN 2880
CBM:            2830 IF TS=64 THEN 2880
BBC:            2830 IF EOF#Y THEN 2880
Apple:          2840 INPUT AU$,BN
MS,CBM:         2840 INPUT#2,AU$,BN
BBC:            2840 INPUT#Y,AU$,BN
CBM:            2841 LET TS=ST
                2845 LET H$=AU$+MID$(BA$,1,20-LEN(AU$))
                2850 IF MID$(R$,32,20)=H$ THEN 2830
Apple:          2860 PRINT D$;"WRITE";"TEMP"
                2861 PRINT AU$:PRINT BN:PRINT D$
MS,CBM:         2860 PRINT#3,AU$:PRINT#3,BN
BBC:            2860 PRINT#Z,AU$:PRINT#Z,BN
                2870 GOTO 2830
Apple:          2879 POKE 216,0
                2880 PRINT D$;"CLOSE";"TEMP"
                2890 PRINT D$;"DELETE";"IXFILE"
                2900 PRINT D$;"RENAME TEMP,IXFILE"
                2910 PRINT D$;"CLOSE";"IXFILE"
MS:             2880 CLOSE 2
                2890 KILL"IXFILE"
                2895 CLOSE 3
                2900 NAME "TEMP" AS "IXFILE"
                2910 CLOSE 2
CBM:            2880 CLOSE 2
                2890 OPEN 4,8,15,"S:IXFILE"
                2900 PRINT#4,"R:IXFILE=TEMP":CLOSE 4
                2910 CLOSE 2
BBC:            2880 CLOSE#Y
                2890 *DELETE IXFILE
                2900 *RENAME TEMP IXFILE
                2910 CLOSE#Y
Apple:          2920 GOTO 2070
Others:         2920 RETURN

                2950 REM=====OPEN TEMP FOR OUTPUT=====
Apple:          2960 PRINT D$;"OPEN";"TEMP"
MS:             2960 OPEN "TEMP" FOR OUTPUT AS#3
CBM:            2960 OPEN 3,8,4,"TEMP",S,W
                2961 LET TS=0
BBC:            2960 Z=OPENOUT"TEMP"
                2970 RETURN
```

6.4 Printing out the Unsorted Book Catalogue

With this program the index file is not needed. All records in the main file BFILE which are not empty are printed out. Again, the program is to a large extent identical to that of the last chapter.

```
3000 REM=====PRINT BOOK FILE UNSORTED=====
3005 GOSUB 800:REM OPEN BFILE
3010 GOSUB 500
3020 PRINT"BOOK FILE"
3030 PRINT"=========="
3040 FOR PN=1 TO 99
3050 GOSUB 2500:REM READ RECORD FROM BFILE TO R$
3060 IF LEFT$(R$,1)="E" THEN 3080
3070 GOSUB 3200:REM DISPLAY RECORD ON SCREEN
3080 NEXT PN
3090 INPUT"CONTINUE(Y/N)";A$
3100 IF A$<>"Y"THEN 3090
3105 GOSUB 900:REM CLOSE BFILE
3110 RETURN

3200 REM=====DISPLAY RECORD ON SCREEN===
     ==
3210 LET T$=MID$(R$,2,30)
3220 LET AU$=MID$(R$,32,20)
3230 LET L$=MID$(R$,52,3)
3240 PRINT"BOOK#";PN
3250 PRINT T$:PRINT AU$:PRINT L$
3260 PRINT:PRINT
3270 RETURN
```

If you would like printing to stop when the screen is full, you can incorporate a counter:

```
3003 LET COUNT=0
3270 LET COUNT=COUNT+1
3280 IF COUNT<4 THEN RETURN
3290 LET COUNT=0
3300 INPUT"TYPE >RETURN< TO CONTINUE";RT$
3310 RETURN
```

Now you can look at the contents of four records at your leisure, and then call up the next records by pressing the RETURN key.

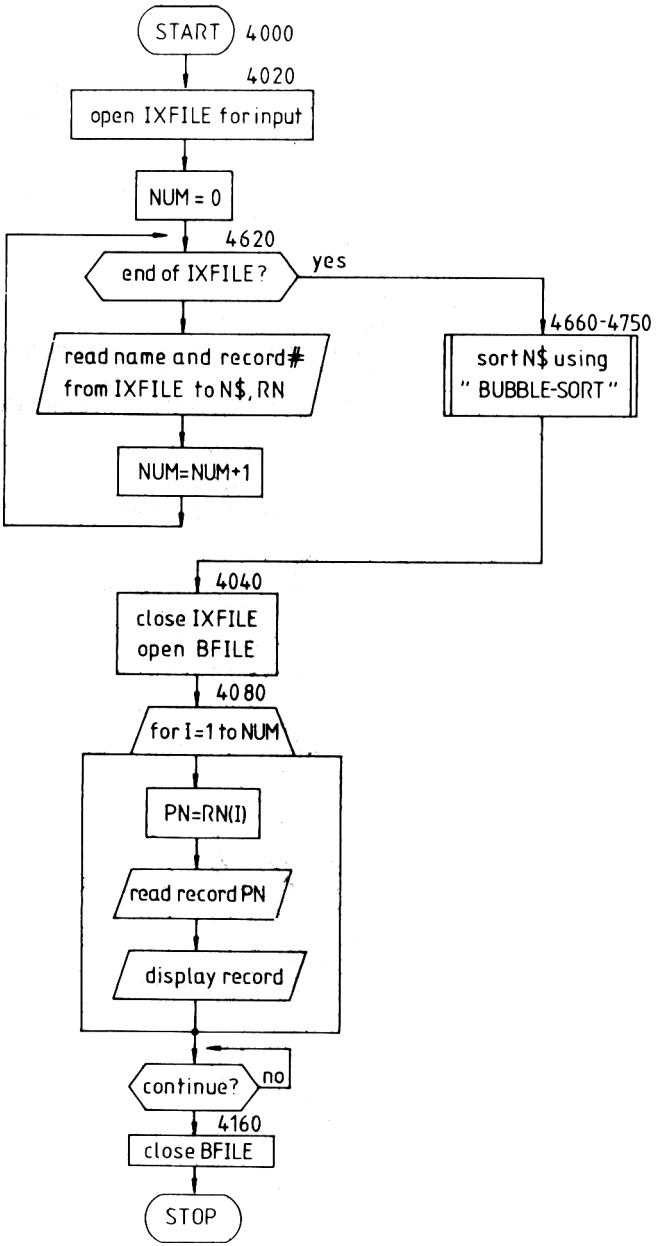


Fig.6.6; flow chart: sorting

6.5 Printing out the Sorted Book Catalogue

The following program prints out the book catalog sorted according to author. Figure 6.6 shows the process in a flow chart.

The file IXFILE is opened for reading. The subroutine for sorting begins at line 4600. The name is read from the index file to N$, and the record number to RN.

Subsequently, the array N$ is sorted alphabetically according to the **bubble sorting method**. If you wish, you can of course use a different sorting method. In the case of large amounts of data, the **quick sort technique** can be recommended. We won't go into details of the sorting algorithm here, but please note that the array RN must also be sorted parallel to the array N$.

When the sorting is finished, the index file is closed and the main file opened. As the number of records is contained in the variable NUM, you can work with a FOR-loop. You assign the record number to the variable PN in each case. The subroutines 2500 and 3200 make sure that the right record is read and displayed. Figure 6.7 shows an example of how arrays N$ and RN look before and after sorting.

Please note that the sorted index file is only to be found in array N$, and **not** in the file IXFILE. It's true, we could store the sorted file instead of the original unsorted one, but in this case the file would be unsorted again after the next new entry.

```
                4000 REM=====PRINT SORTED=====
                4010 GOSUB 500
                4020 GOSUB 4400:REM OPEN IXFILE FOR INPUT
Apple:          4021 PRINT D$;"READ",I$
                4030 GOSUB 4600:REM SORT
                4040 GOSUB 2700:REM CLOSE IXFILE
                4050 GOSUB 800: REM OPEN BFILE
                4060 PRINT"BOOK FILE SORTED"
                4070 PRINT"================"
                4080 FOR I=1 TO NUM:REM NUM=NUMBER OF BOOKS
                4090 LET PN=RN(I)
                4100 GOSUB 2500:REM READ RECORD
                4110 GOSUB 3200:REM DISPLAY RECORD ON SCREEN
                4120 NEXT I
                4130 PRINT:PRINT
                4140 INPUT"CONTINUE(Y/N)";A$
                4150 IF A$<>"Y"THEN 4140
                4160 GOSUB 900:REM CLOSE BFILE
Apple:          4170 GOTO 80
Others:         4170 RETURN

                4400 REM=====OPEN IXFILE FOR INPUT=====
Apple:          4410 PRINT D$;"OPEN",I$
MS:             4410 OPEN I$ FOR INPUT AS#2
```

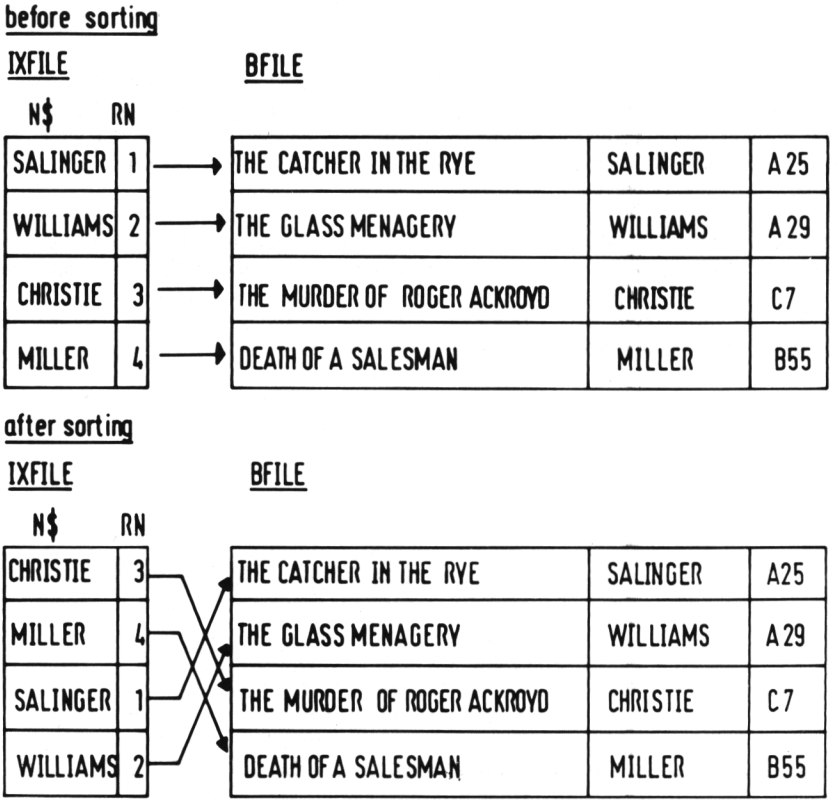


Fig.6.7; Examples in sorting

CBM:	4410 OPEN 2,8,3,I$+",S,R"
BBC:	4410 Y=OPENIN I$
	4420 RETURN
	4600 REM=====BUBBLE-SORT=====
	4610 LET NUM=0:REM COUNT BOOKS
Apple:	4620 ONERR GOTO 4659
MS:	4620 IF EOF(2) THEN 4660
CBM:	4620 IF ST=64 THEN 4660
BBC:	4620 IF EOF#Y THEN 4660
Apple:	4630 INPUT N$(NUM+1),RN(NUM+1)
MS,CBM:	4630 INPUT#2,N$(NUM+1),RN(NUM+1)
BBC:	4630 INPUT#Y,N$(NUM+1),RN(NUM+1)
	4640 LET NUM=NUM+1
	4650 GOTO 4620
Apple:	4659 POKE 216,0
	4660 FOR B=1 TO NUM-1
	4670 FOR C=1 TO NUM-B
	4680 IF N$(C+1)>=N$(C) THEN 4730
	4690 REM SWAP
	4700 H$=N$(C):H=RN(C)
	4710 N$(C)=N$(C+1):RN(C)=RN(C+1)
	4720 N$(C+1)=H$:RN(C+1)=H
	4730 NEXT C
	4740 NEXT B
Apple:	4750 GOTO 4040
Others:	4750 RETURN

6.6 Finding Books by Author

This part of the program searches for all the books by one author. Access to the data stored on one book is gained in two steps. First the index file is searched sequentially for the name of the author. Each time the name is found, name and number are stored in the arrays N$ and RN respectively. A counter BCOUNT counts the number of books found.

Then the data on each book is read from the file BFILE and displayed.

Let's now look at the program in detail. In line 5020 the name of the author to be found is read to AN$. Then a record from the index file is read and compared with AN$ (line 5070). If these are identical, the name and record number are read to the arrays N$ and RN, exactly as in the last program. The counter BCOUNT is incremented by one in line 5100.

When the end of the file has been reached, the index file is closed (line 5120). The counter BCOUNT says whether the author has been found at all, and if so, how many books by this author are stored. In line 5170 the random file is opened. The variable PN,

containing the record number of the record to be read in subroutine 2500, takes on all the record numbers contained in the array RN one after another (line 5190). As usual, the subroutine 3200 displays on the screen the contents of the record read.

```
5000 REM=====SEARCH FOR AUTHOR=====
5010 GOSUB 500
5020 INPUT"AUTHOR'S NAME";AN$
5030 GOSUB 4400:REM OPEN IXFILE FOR INPUT
Apple: 5031 PRINT D$;"READ";I$
5040 LET BCOUNT=0:REM COUNT BOOKS OF SAME
     AUTHOR
Apple: 5050 ONERR GOTO 5119
MS:    5050 IF EOF(2) THEN 5120
CBM:   5050 IF ST=64 THEN 5120
BBC:   5050 IF EOF#Y THEN 5120
Apple: 5060 INPUT AU$,N1
MS,CBM: 5060 INPUT#2,AU$,N1
BBC:   5060 INPUT#Y,AU$,N1
5070 IF AU$<>AN$ THEN 5050
5080 LET N$(BCOUNT+1)=AU$
5090 LET RN(BCOUNT+1)=N1
5100 LET BCOUNT=BCOUNT+1
5110 GOTO 5050
Apple: 5119 POKE 216,0
5120 GOSUB 2700:REM CLOSE IXFILE
5130 IF BCOUNT>0 THEN 5160
5140 PRINT"AUTHOR NOT FOUND"
5150 GOTO 5230
5160 PRINT"AUTHOR FOUND":PRINT
5170 GOSUB 800:REM OPEN BFILE
5180 FOR I=1 TO BCOUNT
5190 LET PN=RN(I)
5200 GOSUB 2500:REM READ RECORD
5210 GOSUB 3200:REM DISPLAY RECORD ON SCREEN
5220 NEXT I
5230 INPUT"CONTINUE(Y/N)";A$
5240 IF A$<>"Y"THEN 5230
5250 GOSUB 900:REM CLOSE BFILE
Apple: 5260 GOTO 80
Others: 5260 RETURN
6000 END
```

Note

If only one book is stored, some computers (e.g. VIC 20) will produce an error in the subroutine "print sorted", because they execute the loop in line 4660 once, whereas they shouldn't do so at all.

Chapter 7

Linked Lists

7.1 Principles

In this chapter, we will be looking at a file whose records are stored in the manner of a linked list. This form of storing is an alternative to the sorted sequential and index sequential method. It has the advantage that, when a record is inserted, the other records **don't have to be moved**. Again, like index sequential files, the file can be sorted according to several keys. The disadvantages are that one or more pointers have to be stored in each record, that there is no direct access, and that programming with linked lists isn't exactly simple. Many an experienced programmer shies away from the idea - possibly, then, a challenge for you to tackle the subject.

Have a look at Figure 7.1; there you see an example of a linked list. The **header** points to the start or head of the list. In our example, the list starts with record No.5. Each record contains an **information section** and a **pointer to the next record**. Let's first suppose that the information section merely consists of a name. The whole list should be in alphabetical order in respect of these names. The pointer of BAKER carries the value 9, as the successor is to be found in the 9th record. This is DAVIS, and in its turn has a pointer which points to SPENCER in the 3rd record. As SPENCER has no successor, we set the pointer to zero.

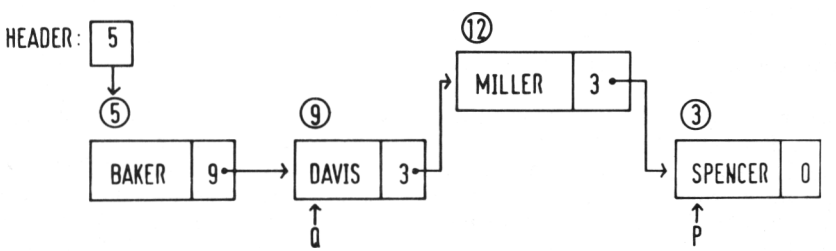


Fig.7.1; example of a linked list

The example shows an essential point: in linked lists, the individual records **need not be arranged next to one another**, but can in principle be scattered where you like. The relationship of their order is established by pointers.

How do you insert a record into a linked list? We'll take the name MILLER as an example, to be inserted between DAVIS and SPENCER (see Figure 7.2). Let us suppose the next free record is Number 12. We write the name MILLER into this record. Then we search the list sequentially by scanning it with two pointers P and Q. At the beginning Q=5 and P=9. We check whether the name in record P is alphabetically greater than the name to be inserted.

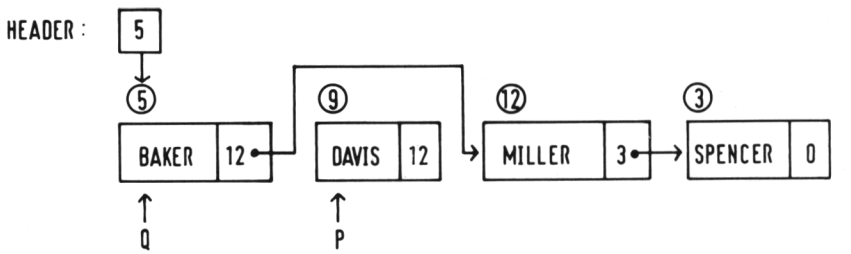


Fig. 7.2; inserting a record into a linked list

At first, this is not the case, as DAVIS is alphabetically less than MILLER. We move the pointers on one position: Q is now pointing to the 9th record, P to the 3rd. These positions are shown in Figure 7.2.

This time, the name in record P is greater than the one to be inserted: SPENCER > MILLER. Now the pointer in record Q is altered; instead of the number 3, the number 12 is entered. Also, the pointer at MILLER is set to 3. This cuts open the chain between DAVIS and SPENCER. The best thing for you to do in order to get used to the pointer method would be to insert further names on a sheet of paper. Try, for example, the name CARTER, by choosing any free record number for CARTER and inserting this new record between BAKER and DAVIS.

Inserting at the beginning of the list requires an alteration to the header; inserting at the end is done when a pointer with the value 0 is met. We'll come back to these two particular cases in our program example.

Deleting a record is done by removing it from the chain. Figure 7.3 shows how the name DAVIS is removed from the list. Again, we use two pointers P and Q. If the name in record P is the same as the one to be deleted, then this record is "bracketed off". This is done by inserting in record Q the pointer which up to now was in record P (in the example: 12). The pointer of BAKER thus points to the successor to DAVIS, which is MILLER in record 12. As there is no longer any pointer leading to DAVIS, this has become what is called "garbage". At particular intervals you can call up a "garbage collector", which releases this record for further use.

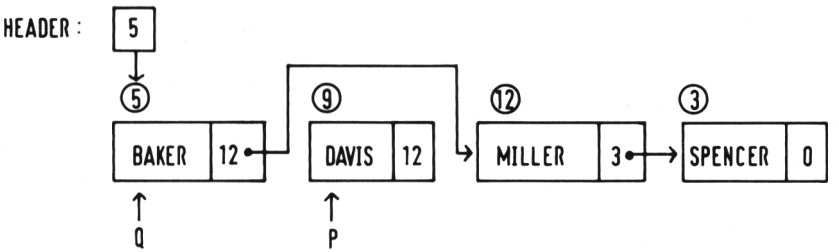


Fig. 7.3; deleting a record in a linked list

Searching for a record is done sequentially; you simply run down the chain. The same goes for printing out. If you want to sort the list according to other keys, you have to construct **sub chains**. We'll come back to this in a moment in our applied example.

7.2 Example: A Customer Index

For our example of a linked list let's look at the customer index of a car dealer. We assume that he sells three makes of car, which we'll call A, B and C. Each customer clearly belongs to one of the groups A, B or C. Occasionally, the dealer will want to send letters to either a particular group of customers, or to all of them. With a sequential file he would always have to search the whole file, even if he only wanted to print out the names of the customers in group A. With our file we will construct three sub chains linking the customers of each group.

Look at Figure 7.4. It shows an example of nine customers, divided into the three groups A, B and C. Each group list is in alphabetical order, and the first empty record has the number 3.

group A:

name	record#
ADAMS	9
CARTER	3
SMITH	4
SPENCER	8

group B:

name	record#
ABEL	5
BAKER	7
MILLER	11

group C:

name	record#
DAVIS	6
WINFIELD	10

Fig.7.4; example of group distribution with nine customers

You could now create three different sorted sequential files, but this would have the disadvantage that you could not print out the complete customer index in alphabetical order. Besides, you might have to move a lot of records when inserting or deleting.

For this reason we choose a linked storage, which has a sub chain for each group of customers. Each record contains two pointers in addition to the information. In Figure 7.5, the nine records are listed according to the record number. The figure marked with an arrow is the header of each list. In all, there are four headers:

Header L: Header of the complete list

Header A: Header of the group list A

Header B: Header of the group list B

Header C: Header of the group list C

record#	name	group	pointer 1	pointer 2
3	CARTER	A	6	4
4	SMITH	A	8	8
L, B →5	ABEL	B	9	7
C →6	DAVIS	C	11	10
7	BAKER	B	3	11
8	SPENCER	A	10	2
A ⟶9	ADAMS	A	7	3
10	WINFIELD	C	2	2
11	MILLER	B	4	2

Fig. 7.5; table diagram of the linked list

Compare the Figures 7.4 and 7.5. For example, the group list C begins with the 6th record, containing the name DAVIS. The first pointer points to the 11th record, namely to MILLER, as this follows DAVIS alphabetically. The second pointer points to the 10th record. There we find the name WINFIELD; this is the successor of the name DAVIS when we look at group C. Analyse the other two part chains in the same way. Please note that the record with the number 2 marks the end of the list. A more exact explanation is given below.

In Figure 7.6 you can see the structure of a record. We reserve 20 bytes for the name and 35 for the address. We'll agree to enter the address in the following form:

street/town or city

e.g: 37 BERRINGTON RD./NORWICH

We'll use one byte for the group label and three bytes for each of the two pointers. This, of course, depends on the maximum possible length of the file.

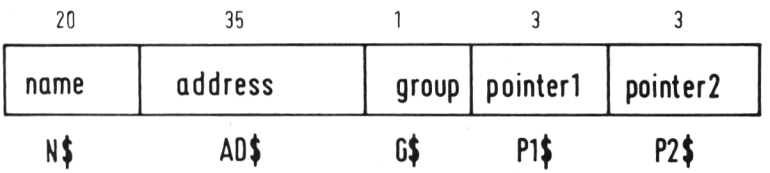


Fig.7.6; structure of a record

Thus every record has a length of 62 bytes; with the additional RETURN symbol, 63 bytes. In our example program we'll restrict ourselves to a maximum of 98 records. The file is given the name CLFILE (client file). The program is to perform the following tasks:

1. Initializing the file CLFILE
2. Entering new customers
3. Deleting customers' names
4. Printing the contents of CLFILE
5. Printing the group lists A, B or C
6. Finding the name of a customer

We'll give the program the name CLIENT.

In Figure 7.7 you will see the structure diagram. You remember the modular structure. Using central subroutines, particularly for reading, writing and displaying a record, makes the program clearer.

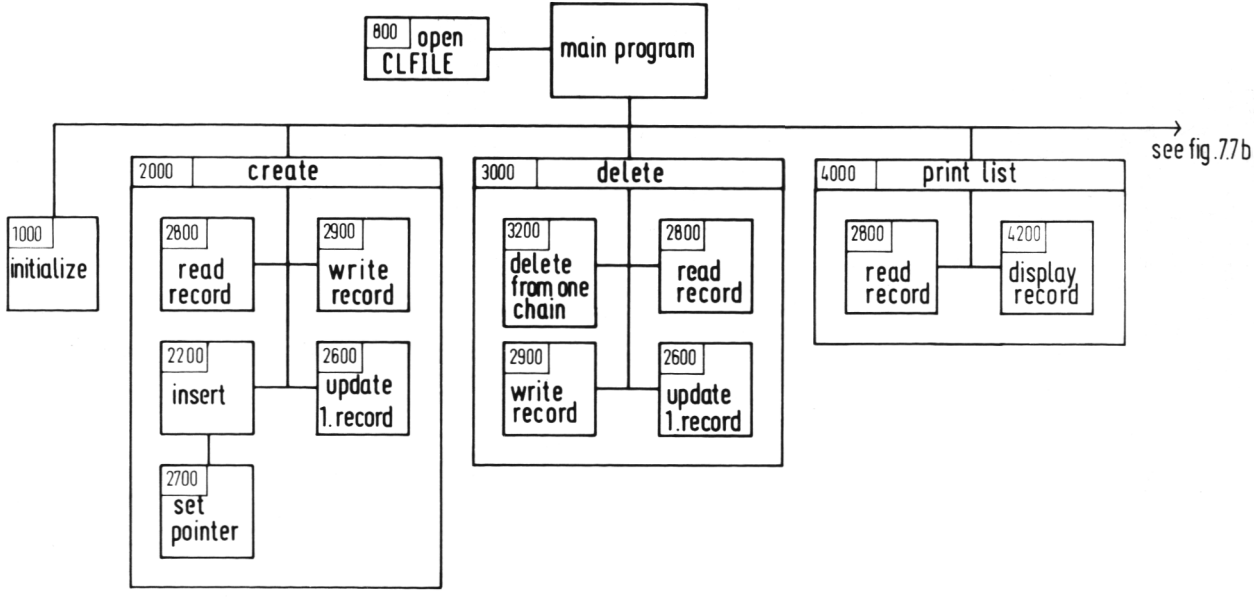


Fig.7.7a; structure diagram for the program CLIENT

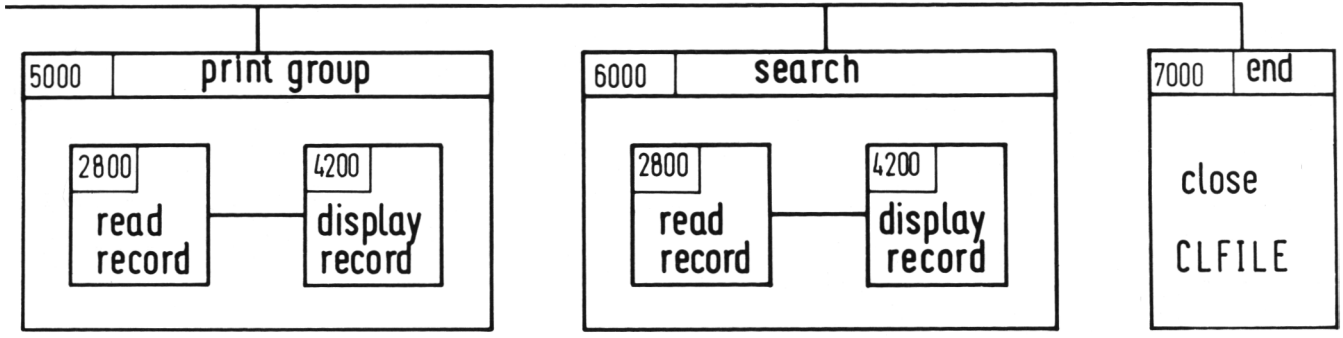


Fig.7.7b; structure diagram for the program CLIENT

The **first record** plays a special part. The four headers and the next free position are stored in it. At the very beginning, all the pointers are pointing to the 2nd record. We write the hexadecimal value FF to this record when initializing. As this value is greater than any other name, the 2nd record always stands at the end of the list. When searching the list, we know when we've come to the end when we read the 2nd record. In Figure 7.8 you can see the contents of the first record when the list is empty. The next free position is Number 3.

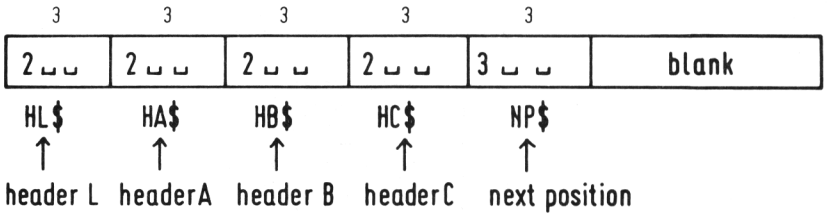


Fig. 7.8; contents of the first record at the start

Note

We had some difficulty with our VIC 10. First it didn't accept the record length of 63 in line 810, so we used 64. Again, the PRINT# statement in line 2911 didn't always work correctly, so we inserted a waiting loop in line 2912. Perhaps these programming tricks will help you when writing your own programs.

There follows the program menu, constructed as usual. The two variables BN$ and BA$ contain 20 and 35 blanks respectively to fill up name and address.

Apple:

```
10 REM=====CLIENT FILE=====
15 DIM MDR(3)
20 LET F$="CLFILE"
21 LET D$=CHR$(4)
30 LET BP$=CHR$(7):REM BEEP
40 LET BN$="                    ":REM 20 BLANKS
50 LET  BA$="                                   ":REM
   35 BLANKS
60 GOSUB 800:REM OPEN CLFILE
70 GOSUB 500
80 PRINT TAB(10);"CLIENT FILE"
90 PRINT:PRINT
100 PRINT TAB(10);"1=INITIALIZE CLIENT FILE"
110 PRINT TAB(10);"2=CREATE NEW ENTRY"
120 PRINT TAB(10);"3=DELETE CLIENT FROM CLFILE"
130 PRINT TAB(10);"4=PRINT CLFILE"
140 PRINT TAB(10);"5=PRINT GROUP"
150 PRINT TAB(10);"6=SEARCH FOR CLIENT"
160 PRINT TAB(10);"7=END"
170 PRINT:PRINT
```

```
180 PRINT TAB(10);"SELECT NUMBER,PLEASE"
190 INPUT N
200 IF N<1 OR N>7 THEN PRINT BP$:GOTO 190
210 ON N GOSUB 1000,2000,3000,4000,5000,6000,7000
220 GOTO 70

500 REM=====CLEAR SCREEN=====
```

Apple:
```
510 HOME
```

MS,BBC:
```
510 CLS
```

CBM:
```
510 PRINT CHR$(147)
520 RETURN

800 REM=====OPEN CLFILE=====
```

Apple:
```
810 PRINT D$;"OPEN";F$;",L63"
```

MS:
```
810 OPEN F$ AS#1 LEN=62
811 FIELD#1,62 AS RC$
```

CBM:
```
810 OPEN 1,8,2,F$+",L,"+CHR$(64)
811 OPEN 3,8,15
812 PRINT#3,"P"+CHR$(2)+CHR$(100)+CHR$(0)+CH
    R$(1)
813 PRINT#1,CHR$(255)
```

BBC:
```
810 X=OPENUP F$
820 RETURN
```

7.3 Initializing the File CLFILE

Records 2 - 99 are initialized as follows:

Byte 1: 'FF' (decimal 255)
Bytes 2-56: blank
Bytes 57-62:000000

The first record is intialized as shown in Figure 7.8.

```
1000 REM=====INITIALIZE CLFILE=====
1010 GOSUB 500
1020 PRINT"CLFILE WILL BE DELETED"
1030 INPUT"TYPE Y TO CONFIRM";A$
1040 IF A$<>"Y"THEN RETURN
```

BBC:
```
1045 CLOSE#X:X=OPENOUT F$
1050 LET C$=CHR$(255)
1060 FOR I=1 TO 55:LET C$=C$+" ":NEXT I
1070 LET C$=C$+"000000"
1080 FOR I=2 TO 99
```

Apple:
```
1090 PRINT D$;"WRITE";F$;",R";I
1091 PRINT C$
```

```
MS:          1090 LSET RC$=C$
             1091 PUT#1,I
CBM:         1090 PRINT#3,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(1)
             1091 PRINT#1,C$
BBC:         1090 PTR#X=64*I
             1091 PRINT#X,C$
             1100 NEXT I
Apple:       1101 PRINT D$
             1110 LET C$="2  2  2  2  3"
             1120 FOR I=1 TO 47:LET C$=C$+" ":NEXT I
Apple:       1130 PRINT D$,"WRITE",F$,",R1"
             1131 PRINT C$
             1132 PRINT D$
MS:          1130 LSET RC$=C$
             1131 PUT#1,1
CBM:         1130 PRINT#3,"P"+CHR$(2)+CHR$(1)+CHR$(0)+CHR$(1)
             1131 PRINT#1,C$
BBC:         1130 PTR#X=64
             1131 PRINT#X,C$
             1140 RETURN
```

7.4 Inserting the Name of a new Customer

Let us have a detailed look at inserting a record into a linked list. For you really to understand the following program, we'll have to consider different situations.

1. Inserting the first record into the empty list.

The first name we are going to insert is customer CARTER, who belongs to group A. We read the 1st record, containing the header L and the next free position. Both variables have the following value:

Header L: 2
Next position: 3

Figure 7.9 shows the empty list. It contains only record No. 2, which serves as end marker.

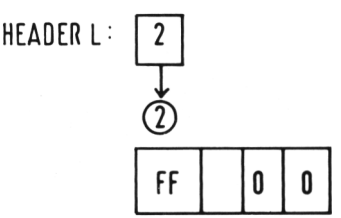


Fig.7.9; the empty list

We now "cut open" the connection between header L and record 2, and put the new record with the number 3 in between. Figure 7.10 shows the result:

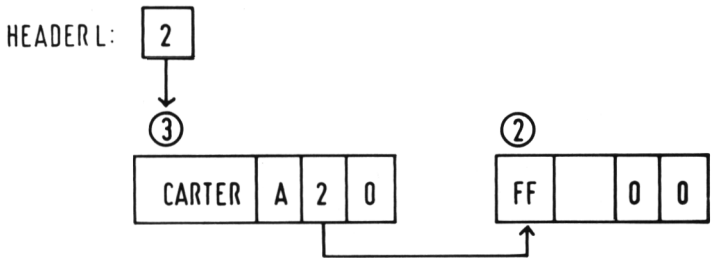


Fig.7.10; Inserting the first record

For the sake of simplicity, we have left out the address and only put in the name.

Now CARTER has been entered in the list, the sub chain for group A must be constructed. This is done on the same principle. Figure 7.11 shows the result:

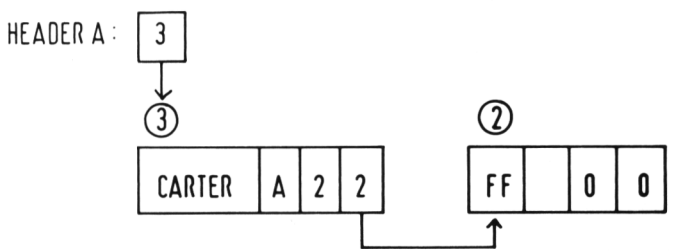


Fig.7.11; construction of the sub chain for group A

After that, the pointer to the next position is increased by one. The following values of the 1st record have changed:

Header L: 3
Header A: 3
Next position: 4

These altered values are stored in the first record.

2. Inserting the next record at the end

The next record is SMITH (see Figure 7.5). This is entered after CARTER. Figure 7.12 shows the result:

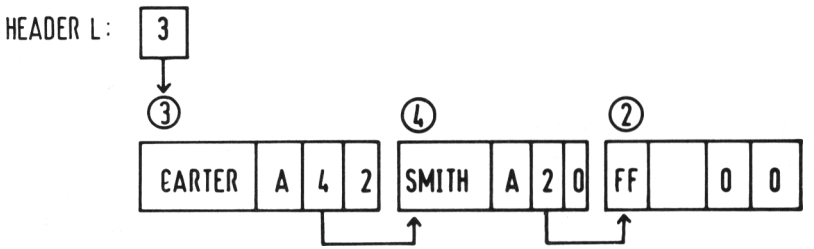


Fig.7.12; the list contains two records

As SMITH also belongs to group A, his name is appended to CARTER in the sub chain for group A. Please note Figure 7.13:

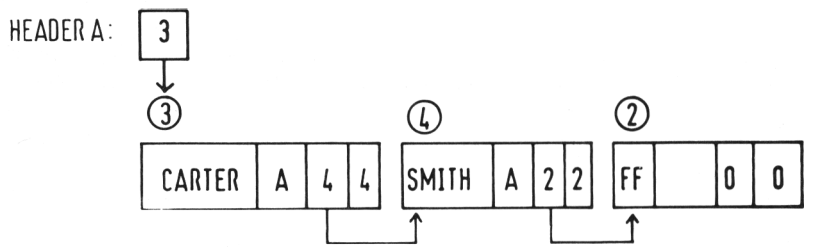


Fig.7.13; Inserting into the sub chain for group A

3. Inserting the next record at the beginning

The next customer is ABEL; his name must be inserted in the list before CARTER. Figure 7.14 shows the new list:

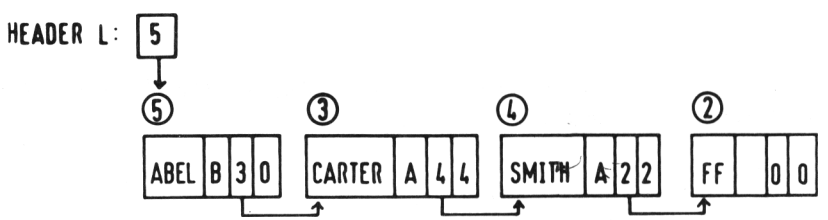


Fig.7.14; List with three records

From the illustrations you can see how the 2nd record is pushed further and further towards the right-hand edge. As ABEL belongs to group B, header B must now be altered:

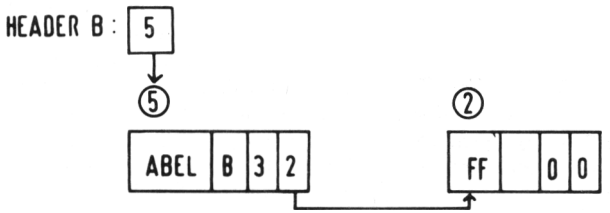


Fig.7.15; construction of the sub chain for group B

4. Entering further names in the middle

In most cases, a new record is inserted somewhere in the middle of the list. The next name, DAVIS, comes between CARTER and SMITH, for example. As it belongs to group C, header C points to this record. In Figure 7.16, all the pointers are shown. As you see, even with only five records it is beginning to look complicated. You should, however, take the trouble to follow and to understand each individual pointer. The best thing for you to do would be to get an enormous sheet of paper and build up our example list from Figure 7.5 again bit by bit!

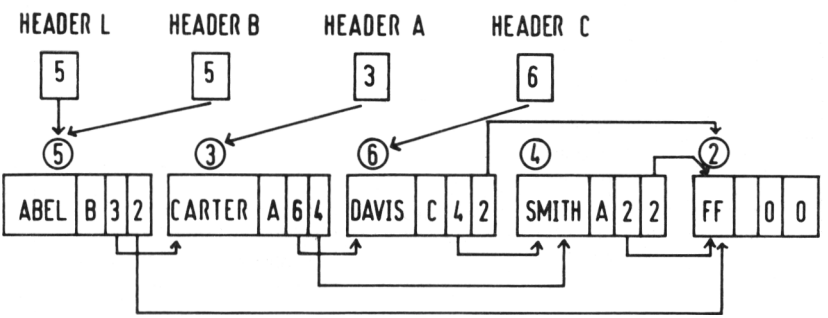


Fig 7.16 all pointers of the list with five records

The different case examples have shown you the following: we need a subroutine which will insert a new record in the right place in the list. This is called up **twice**; firstly for inserting into the main list, and secondly for inserting into the sub list. Using a parameter, which we'll call **PARA**, we tell the program which list it should process:

PARA=0 : Main list
PARA=1 : Sub list A
PARA=3 : Sub list B
PARA=4 : Sub list C

Now have a look at the flow chart in Figure 7.17. It shows the subroutine for inserting a record, but doesn't go into detail. The first record is read to get the values for Header and Next position. If the name is alphabetically smaller than the name of the first record in the list, we branch to line 2360. There the header is altered. If this is not the case, the pointer Q is pointing now to the first record in the list, while P points to the following record. The whole list is searched with the pair of pointers Q and P. As long as the new name is larger than or equal to the name in record P, both pointers are shifted one position. At the end of the list at the latest, the condition in line 2330 is fulfilled. Then the pointers in records Q and NP are set accordingly. The new name is written to the record NP and the first record is updated.

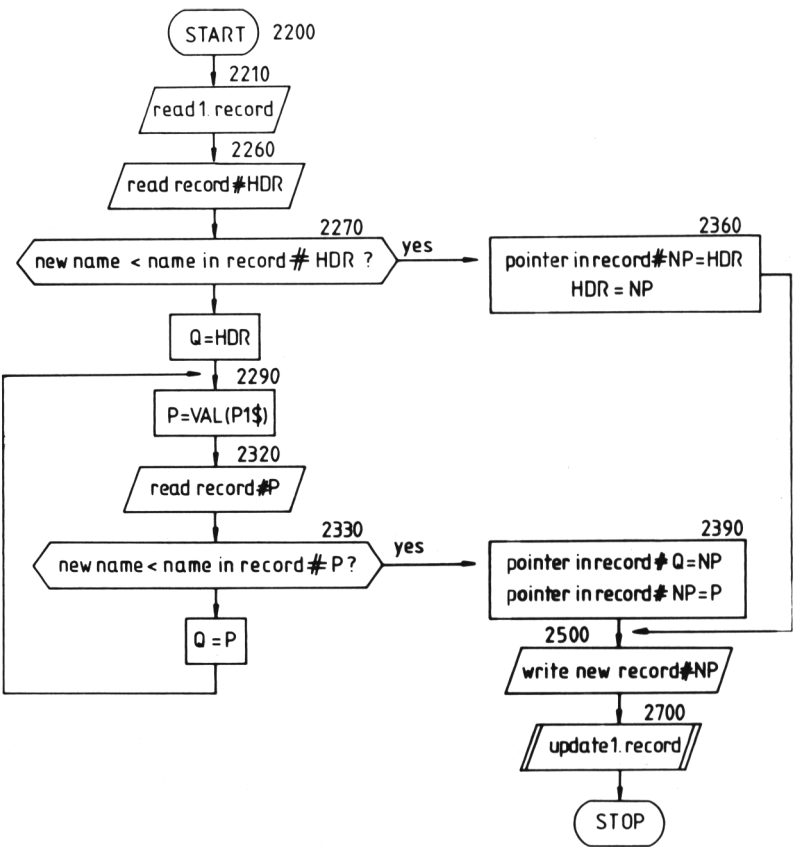


Fig.7.17; flow chart for inserting a record

```
2000 REM=====CREATE NEW ENTRY=====
2010 GOSUB 500
2020 INPUT"NAME";N$
2030 INPUT"ADDRESS";AD$
2040 INPUT"GROUP(A,B,C)";G$
2050 LET PARA=0:REM INSERT INTO MAIN LIST
2060 GOSUB 2200:REM INSERT NEW RECORD
2070 GOSUB 2600:REM UPDATE 1.RECORD
2080 LET PARA=ASC(G$)-64
2090 GOSUB 2200:REM INSERT INTO GROUP LIST
2100 LET NP=NP+1:REM UPDATE NEXT POSITION POINTER
2110 GOSUB 2600:REM UPDATE 1.RECORD
CBM:  2111 CLOSE 1:CLOSE 3:GOSUB 800
2120 RETURN
2200 REM=====INSERT NEW RECORD=====
2210 LET RN=1:GOSUB 2800:REM 1.RECORD
2220 FOR I=0 TO 3:REM READ HEADER
2230 LET HDR(I)=VAL(MID$(R$,3*I+1,3))
2240 NEXT I
2250 LET NP=VAL(MID$(R$,13,3)):REM NEXT POSITION
2260 LET RN=HDR(PARA):GOSUB 2800:REM READ RECORD
2270 IF N$<MID$(R$,1,20)THEN 2360
2280 LET Q=HDR(PARA)
2290 IF PARA=0 THEN 2310
2300 LET P=VAL(MID$(R$,60,3)):GOTO 2320
2310 LET P=VAL(MID$(R$,57,3))
2320 LET RN=P:GOSUB 2800:REM RECORD  P
2330 IF N$<MID$(R$,1,20)THEN 2390
2340 LET Q=P
2350 GOTO 2290
2360 LET P1$=STR$(HDR(PARA))
MS,CBM:  2361 LET P1$=RIGHT$(P1$,LEN(P1$)-1)
2370 LET HDR(PARA)=NP
2380 GOTO 2420
2390 LET RN=Q:GOSUB 2800:REM READ RECORD#Q
2400 GOSUB 2700:REM ADD POINTER TO R$,WRITE RECORD#Q
2410 LET P1$=STR$(P)
MS,CBM:  2411 LET P1$=RIGHT$(P1$,LEN(P1$)-1)
2420 LET P1$=P1$+MID$("   ",1,3-LEN(P1$))
2430 IF PARA=0 THEN 2460
2440 LET RN=NP:GOSUB 2800:REM READ NEW RECORD#NP
2450 LET R$=LEFT$(R$,59)+P1$:GOTO 2500
2460 LET R$=N$+MID$(BN$,1,20-LEN(N$))
2470 LET R$=R$+AD$+MID$(BA$,1,35-LEN(AD$))+G$
2480 LET S$=LEFT$(R$,56)+P1$+"000"
2490 LET R$=S$
2500 LET RN=NP:GOSUB 2900:REM WRITE RECORD#NP
2510 RETURN
2600 REM=====UPDATE 1.RECORD
2610 LET HL$=STR$(HDR(0))
MS,CBM:  2615 LET HL$=RIGHT$(HL$,LEN(HL$)-1)
2618 LET R$=HL$+MID$("   ",1,3-LEN(HL$))
2620 LET HA$=STR$(HDR(1))
MS,CBM:  2625 LET HA$=RIGHT$(HA$,LEN(HA$)-1)
```

```
2628 LET R$=R$+HA$+MID$("  ",1,3-LEN(HA$))
2630 LET HB$=STR$(HDR(2))
MS,CBM:
2635 LET HB$=RIGHT$(HB$,LEN(HB$)-1)
2638 LET R$=R$+HB$+MID$("  ",1,3-LEN(HB$))
2640 LET HC$=STR$(HDR(3))
MS,CBM:
2645 LET HC$=RIGHT$(HC$,LEN(HC$)-1)
2648 LET R$=R$+HC$+MID$("  ",1,3-LEN(HC$))
2650 LET NP$=STR$(NP)
MS,CBM:
2655 LET NP$=RIGHT$(NP$,LEN(NP$)-1)
2660 LET R$=R$+NP$+MID$("  ",1,3-LEN(NP$))
2670 FOR I=1 TO 47:LET R$=R$+" ":NEXT I
2680 LET RN=1:GOSUB 2900
2690 RETURN
2700 REM======ADD POINTER TO R$ AND WRITE RECORD#Q
=====
2710 LET P$=STR$(NP)+MID$("  ",1,3-LEN(STR$(NP)))
MS,CBM:
2710 LET H$=RIGHT$(STR$(NP),LEN(STR$(NP))-1)
2711 LET P$=H$+MID$("  ",1,3-LEN(H$))
2720 IF PARA=0 THEN 2740
2730 LET S$=LEFT$(R$,59)+P$:GOTO 2750
2740 LET S$=LEFT$(R$,56)+P$+MID$(R$,60,3)
2750 LET R$=S$
2760 LET RN=Q:GOSUB 2900
2770 RETURN


2800 REM======READ RECORD======
Apple:
2810 PRINT D$;"READ";F$;",R";RN
2811 INPUT R$
2812 PRINT D$
MS:
2810 GET#1,RN
2811 LET R$=RC$
CBM:
2810 PRINT#3,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
2811 INPUT#1,R$
BBC:
2810 PTR#X=64*RN
2811 INPUT#X,R$
2820 RETURN

2900 REM======WRITE RECORD======
Apple:
2910 PRINT D$;"WRITE";F$;",R";RN
2911 PRINT R$
2912 PRINT D$
MS:
2910 LSET RC$=R$
2911 PUT#1,RN
CBM:
2910 PRINT#3,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
2911 PRINT#1,R$
2912 FOR W=1 TO 1000:NEXT W
BBC:
2910 PTR#X=64*RN
2911 PRINT#X,R$
2920 RETURN
```

Now let's look at the program in detail. We write the line numbers on the left and the explanation on the right.

2020-2040: The new data is read in. We have not checked the length this time.

2050-2060: The subroutine 2200 for inserting a record is called up with the parameter value 0, i.e., inserting into the main list.

2070: The header HL is altered.

2080: The parameter PARA is set to 1,2 or 3, according to whether the group label A,B or C is found in G$.

2090: The new record is inserted into the group list.

2100-2110: The pointer to the next empty position is incremented by one, the first record is altered and written back.

Subroutine 2200

2210: The first record is read.

2220-2240: The four Headers HL,HA,HB and HC are brought into the array HDR.

2250: NP points to the next empty position.

2260: The first record in the list is read.

2270: If the new name is less than the name read, we branch off to 2360.

2280: The pointer Q points to the beginning of the list.

2290-2310: The pointer P is positioned on the record following Q. According to the value of the parameter, either the first or the second pointer from R$ is selected.

2320: The record P is read.

2330: If the new name is less than the one in record P, we branch off to 2390.

2340-2350: The pointer Q moves one position further, then we jump back to 2290.

2360: P1$ is given the header as value.

2370: The Header now points to the new record.

2390-2400: The record Q is read, the pointer P1$ is inserted.

2410-2420: The contents of pointer P1$ are filled up with blanks.

2430-2500: If the parameter has the value 0 (i.e., inserting into the main list), the new record is constructed in R$ and written to the position NP. If the insertion is to be made into a group list, however, the new record is read first, as it has already been written to the main list by the same subroutine.

Subroutine 2600:

2610-2660: The first record is constructed.

2670:
47 blanks are appended.

2680: The first record is written back.

Subroutine 2700:

2710:	The contents of NP (i.e., the number of the next empty record) are transferred to P$ and filled up with blanks.
2720-2750:	The value of P$ is inserted in R$ as first or second pointer.
2760:	R$ is written into record Q.

Subroutines 2800 and 2900 are the same as in the last chapter. The number of the record to be read or written is found in the variable RN.

7.5 Deleting a Customer's Name

In Figure 7.18 you can see the flow chart. We read the first record first, and position the two pointers Q and P accordingly. Then we read record P, and check whether the name to be deleted ND$ is the same as the name in record P. If so, then we either correct the header (Q=1) or alter the pointer in record Q. Subroutine 3200 is then called up again in order to delete the name from the group list.

If the name to be deleted has not been found yet, then the two pointers Q and P are moved on one position. The condition P=2 is fulfilled when the end of the list is reached.

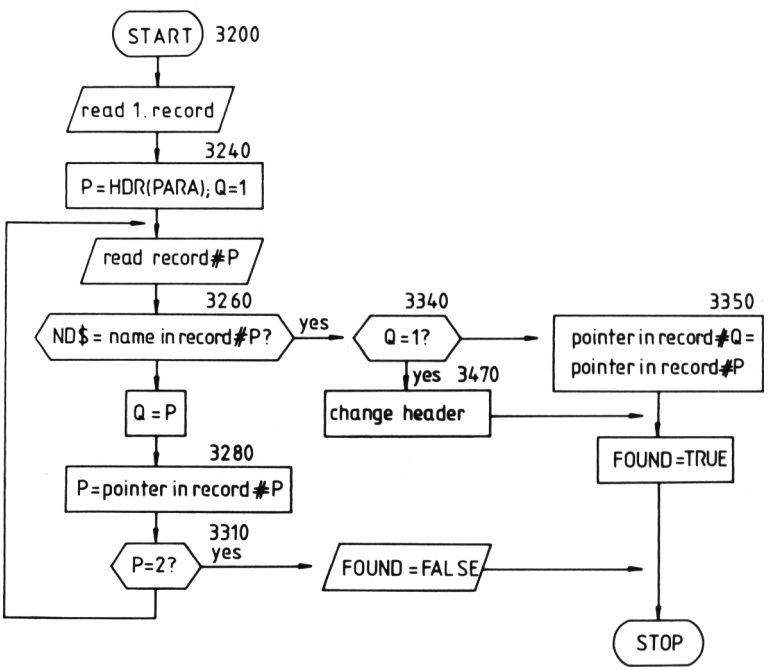


Fig.7.18; flow chart for deleting a record

If you have carefully studied the program for inserting in the last section, you will immediately understand the program for deleting, as it works by the same method. In line 3040, the logical variable FOUND is set to 0 (FALSE). If the name to be deleted is not found at the first attempt, the subroutine in line 3320 sets this variable to FALSE; if the name is found, on TRUE in line 3450 (TRUE=-1). The value of the variable FOUND is tested in line 3060.

The subroutine 3200 uses the same parameter PARA as the previous subroutine 2200. The value of this parameter decides which pointer is used (lines 3280 and 3400). You will understand the function of the program best if you look at the example list in Figure 7.16, and, for example, delete the name CARTER. Go through the program step by step, and alter the pointers accordingly.

```
3000 REM=====DELETE CLIENT=====
3010 GOSUB 500
3020 INPUT"NAME TO DELETE";ND$
3030 LET ND$=ND$+MID$(BN$,1,20-LEN(ND$))
3040 LET PARA=0:LET FOUND=0
3050 GOSUB 3200: REM DELETE RECORD FROM MAIN
     LIST
3060 IF FOUND THEN 3090
3070 PRINT"CLIENT NOT FOUND":PRINT
3080 GOTO 3120
3090 LET PARA=ASC(G$)-64
3100 GOSUB 3200:REM DELETE RECORD FROM GROUP
     LIST
3110 PRINT"CLIENT DELETED":PRINT
3120 INPUT"CONTINUE(Y/N)";A$
3130 IF A$<>"Y"THEN 3120
3140 RETURN

3200 LET RN=1:GOSUB 2800:REM READ 1.RECORD
3210 FOR I=0 TO 3
3220 LET HDR(I)=VAL(MID$(R$,3*I+1,3))
3230 NEXT I
3240 LET P=HDR(PARA):LET Q=1
3250 LET RN=P:GOSUB 2800:REM READ RECORD# P
3260 IF ND$=MID$(R$,1,20) THEN 3340
3270 LET Q=P
3280 IF PARA=0 THEN 3300
3290 LET P=VAL(MID$(R$,60,3)):GOTO 3310
3300 LET P=VAL(MID$(R$,57,3))
3310 IF P<>2 THEN 3250
3320 LET FOUND=0:REM CLIENT NOT FOUND
3330 RETURN
3340 IF Q=1 THEN 3470:REM NEW HEADER
3350 IF PARA=0 THEN 3370
3360 LET P1$=MID$(R$,60,3):GOTO 3380
3370 LET P1$=MID$(R$,57,3)
```

```
3380 LET G$=MID$(R$,56,1)
3390 LET RN=Q:GOSUB 2800:REM READ RECORD#Q
3400 IF PARA=0 THEN 3420
3410 LET S$=LEFT$(R$,59)+P1$:GOTO 3430
3420 LET S$=LEFT$(R$,56)+P1$+RIGHT$(R$,3)
3430 LET R$=S$
3440 LET RN=Q:GOSUB 2900:REM WRITE RECORD#Q
3450 LET FOUND=-1:REM FOUND=TRUE
3460 RETURN
3470 IF PARA=0 THEN 3490
3480 LET P1$=MID$(R$,60,3):GOTO 3500
3490 LET P1$=MID$(R$,57,3)
3500 LET G$=MID$(R$,56,1)
3510 LET HDR(PARA)=VAL(P1$)
3520 GOSUB 2600:REM UPDATE 1.RECORD
3530 GOTO 3450
```

7.6 Printing out the Complete List

The program for printing out is relatively simple. You position the pointer P at the beginning of the list, read a record and print it out. Then P is positioned on the successor (line 4090). The end of the list is reached when P=2. The subroutine 4200 displays the contents of a record on the screen.

```
4000 REM=====PRINT CLFILE=====
4010 GOSUB 500
4020 PRINT"LIST OF ALL CLIENTS"
4030 PRINT"====================="
4040 LET RN=1:GOSUB 2800:REM READ 1.RECORD
4050 LET P=VAL(LEFT$(R$,3)):REM HEADER L
4060 IF P=2 THEN 4110:REM END OF LIST
4070 LET RN=P:GOSUB 2800:REM READ RECORDP
4080 GOSUB 4200:REM DISPLAY RECORD ON SCREEN
4090 LET P=VAL(MID$(R$,57,3)):REM SUCCESSOR
4100 GOTO 4060
4110 INPUT"CONTINUE(Y/N)";A$
4120 IF A$<>"Y"THEN 4110
4130 RETURN

4200 REM=====DISPLAY RECORD ON SCREEN===
==
4210 LET N$=MID$(R$,1,20)
4220 LET AD$=MID$(R$,21,35)
4230 LET G$=MID$(R$,56,1)
4240 FOR I=1 TO 35:REM SPLIT ADDRESS
4250 IF MID$(AD$,I,1)="/"THEN T=I:I=36:NEXTI:I=T:
     GOTO 4290
4260 NEXT I
```

```
4270 PRINT"BAD ADDRESS":REM / IS MISSING
4280 PRINT BP$:RETURN
4290 PRINT N$:PRINT LEFT$(AD$,I-1)
4300 PRINT MID$(AD$,I+1,35-I-1)
4310 PRINT"GROUP";G$
4320 PRINT:PRINT
4330 RETURN
```

7.7 Printing out the Group List

In principle, printing out the group list works in the same way as printing out the complete list, except that the header is different. In line 5080 it is selected from R$. If, for example, you want to print out the list of Group B, you get the following position in the MID$ function:

```
3*(ASC("B")-64)+1 = 7
```

Header B begins in the 7th position (look at Figure 7.8 again).

```
5000 REM=====PRINT GROUP=====
5010 GOSUB 500
5020 INPUT"GROUP";G$
5030 IF G$<>"A" AND G$<>"B" AND G$<>"C" THEN
     PRINT BP$: GOTO 5020
5040 PRINT:PRINT
5050 PRINT"LIST OF CLIENTS - GROUP";G$
5060 PRINT"=======================
     ====="
5070 LET RN=1:GOSUB 2800:REM READ 1.RECORD
5080 LET H$=MID$(R$,3*(ASC(G$)-64)+1,3)
5090 LET P=VAL(H$)
5100: IF P=2 THEN 5150
5110 LET RN=P:GOSUB 2800:REM READ RECORD#P
5120 GOSUB 4200:REM DISPLAY RECORD ON SCREEN
5130 LET P=VAL(MID$(R$,60,3))
5140 GOTO 5100
5150 INPUT"CONTINUE(Y/N)";A$
5160 IF A$<>"Y"THEN 5150
5170 RETURN
```

7.8 Finding a Customer's Name

The name to be found is input and assigned to S$. The first record, and provides the header. As with printing out, you run through the list with the help of the pointer P. The end of the list is reached when P=2. Printing out the data record found is done the subroutine 4200 again.

```
6000 REM=====SEARCH FOR CLIENT=====
6010 GOSUB 500
6020 INPUT"NAME TO SEARCH FOR";S$
6030 LET RN=1:GOSUB 2800:REM READ 1.RECORD
6040 LET S$=S$+MID$(BN$,1,20-LEN(S$))
6050 LET P=VAL (LEFT$(R$,3))
6060 IF P=2 THEN 6140
6070 LET RN=P:GOSUB 2800:REM READ RECORD#P
6080 IF S$=MID$(R$,1,20)THEN 6110
6090 LET P=VAL(MID$(R$,57,3))
6100 GOTO 6060
6110 PRINT"CLIENT FOUND":PRINT
6120 GOSUB 4200:REM DISPLAY RECORD ON SCREEN
6130 GOTO 6150
6140 PRINT"CLIENT NOT FOUND":PRINT
6150 INPUT"CONTINUE(Y/N)";A$
6160 IF A$<>"Y"THEN 6150
6170 RETURN

7000 REM=====CLOSE AND END=====
```

Apple:	7010 PRINT D$;"CLOSE";F$
MS,CBM:	7010 CLOSE 1
BBC:	7010 CLOSE#X

```
7020 END
```

Chapter 8

Files with Scattered Storage

8.1 Example: An Inventory File

Let's have another look at our inventory program from the fifth chapter. There we presumed that the individual articles or parts were numbered rom 1 to 50. This needn't always be the case in practice, though.

We'll assume that the part numbers are in the range from 1 to 999. We are dealing again with about 50 parts, but this time the numbers are **scattered.** The key will again be the number of the part.

Example:

Part 1: 89

Part 2: 390

Part 3: 725

Part 4: 282

If you want to realize an inventory file of this kind using the method applied in chapter 5, you must use a random file with 999 records. This is, however, 95% empty, so you will be wasting an enormous amount of storage space. We will use a better method.

Whenever the entire range of keys is large compared to the actual number of records in the file, the **scattered storage** method is used. It's also called:

- randomizing
- hashing
- key-to-address transformation

Example:

A company with 3000 employees uses a six-digit personnel number from 000000 to 999999. Instead of creating a file with a million records, they use a hash-technique. With a key-to-address transformation, the file is compressed to the necessary length of 3000 records (see Figure 8.1). The expression "address" is another word for "record number".

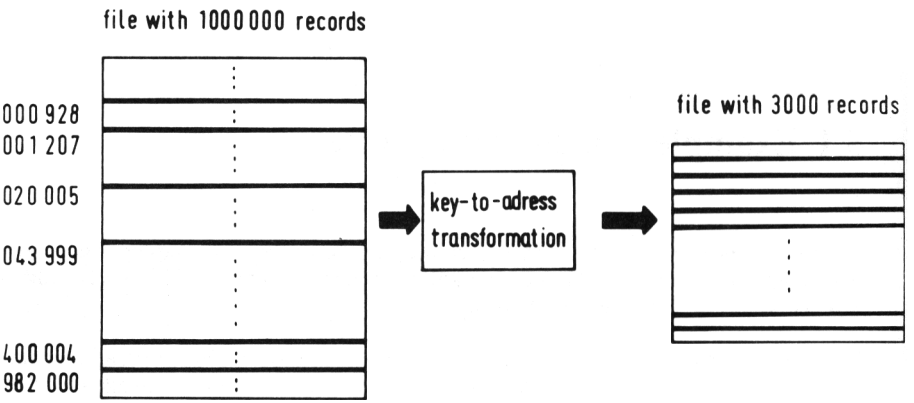


Fig.8.1; example of the use of hash-technique

How is the transformation carried out? A whole range of methods have been developed. These are the four most important:

1. Division: The key is divided by the number of records, and the remainder is the record number.

2. Midsquare: The key is squared and the digits in the middle interpreted as the record number.

3. Folding: The key is taken apart and its components linked in some way, e.g. by adding them.

4. Random: The key is used as input to a random number generator; the product is the record number.

We are going to use the first method in our inventory file. First we select six arbitrary part numbers. Each number is divided by 50, so that values between 0 and 49 will result. As we are beginning the numbering of the records with 1, we will add 1 to the remainder of the division.

Example:

Part Number	Division	Remainder	Remainder+1 = Record Number
27	27 : 50	27	28
158	158 : 50	8	9
170	170 : 50	20	21
458	458 : 50	8	9
599	599 : 50	49	50
970	970 : 50	20	21

The first part with the number 27 is thus stored in the 28th record. The second part goes into the 9th, the third into the 21st. With the fourth part we have what is called a **collision;** it should also go into the 9th record, but this is already occupied.

This example shows you an important point: in key-to-address transformation, collisions can occur if the same address is assigned to more than one key. The two parts numbered 170 and 970 also collide. However, there are also several methods of collision handling, and we are going to use a very simple one: we store all colliding records sequentially in an **overflow file;** this is shown in Figure 8.2.

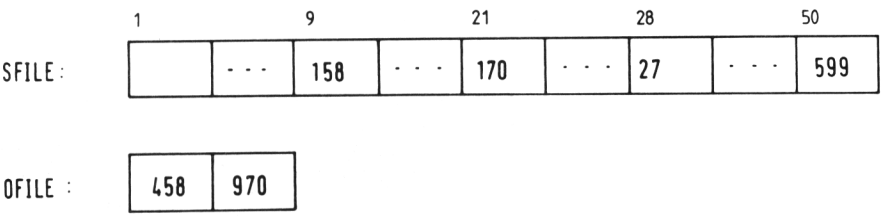


Fig.8.2; example of collision handling

We will call the main file SFILE (scattered storage file) and the overflow file OFILE. First, the three parts with the numbers 27, 158 and 170 are stored in the records 28, 9 and 21 respectively.
The fourth part, number 458, causes a collision with the 9th record, and so it is stored in the overflow file. The next part goes into the 50th record, and the last part, also leading to a collision, is also stored in the overflow file.

The **advantage** of scattered storage lies in the fact that a lot of storage space can be saved. The **disadvantage** becomes clear when a lot of collisions occur. If you are looking for a record stored in the overflow file, you have no direct access to it, and have to search the file sequentially.

Let's now turn to programming our inventory file. For the sake of simplicity we will retain our inventory program from the fifth chapter and adopt its menu and data structures. This offers you two advantages: firstly, you can adopt large parts of the program INVENTORY and save yourself a lot of writing. Secondly, you are already familiar with the problem and can therefore concentrate on what is important, namely the scattered storage.

Figure 8.3 shows an example of the construction of a record. It has the same length as in the fifth chapter (36 bytes). However,the part number must be stored with each part. We will therefore reduce the length of the part name from 25 to 20 bytes and insert a blank between the three figure number and the name.

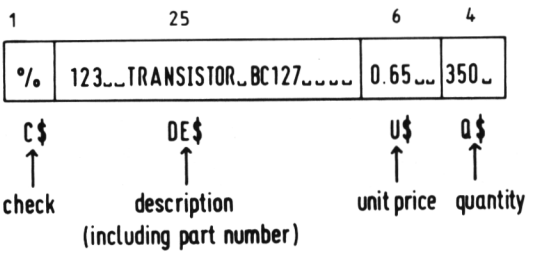


Fig.8.3; example of a record

The following program shows the menu. It is practically identical with the program INVENTORY; all that has to be altered are lines 20 and 40 and some remarks.

Apple:

```
 10 REM=====SCATTERED STORAGE=====
 20 LET F$="SFILE"
 21 LET D$=CHR$(4)
 30 LET BP$=CHR$(7):REM BEEP
 40 LET BD$="                    ":REM 20 BLANKS
 50 LET BU$="      ":REM 6 BLANKS
 60 LET BQ$="    ":REM 4 BLANKS
 70 GOSUB 800:REM OPEN SFILE
 80 GOSUB 500
 90 PRINT TAB(10);"INVENTORY (SCATTERED STORAGE)"
100 PRINT:PRINT
110 PRINT TAB(10);"1=INITIALIZE INVENTORY FILE"
120 PRINT TAB(10);"2=CREATE NEW ENTRY"
130 PRINT TAB(10);"3=PRINT INVENTORY UNSORTED"
140 PRINT TAB(10);"4=ACCESS ONE PART"
150 PRINT TAB(10);"5=END"
160 PRINT:PRINT
170 PRINT TAB(10);"SELECT NUMBER PLEASE"
180 INPUT N
```

```
190 IF N<1 OR N>5 THEN PRINT BP$:GOTO 180
200 ON N GOSUB 1000,2000,3000,4000,5000
210 GOTO 80

500 REM=====CLEAR SCREEN=====
Apple:  510 HOME
MS,BBC: 510 CLS
CBM:    510 PRINT CHR$(147)
        520 RETURN

800 REM=====OPEN SFILE=====
Apple:  810 PRINT D$;"OPEN",F$;",L37"
MS:     810 OPEN F$ AS#1 LEN=26
        811 FIELD#1,36 AS RC$
CBM:    810 OPEN 1,8,2,F$+",L,"+CHR$(37)
        811 OPEN 3,8,15
        812 PRINT#3,"P"+CHR$(2)+CHR$(51)+CHR$(0)+
            CHR$(1)
        813 PRINT#1,CHR$(255)
BBC:    810 X=OPENUP F$
        820 RETURN
```

8.2 Initializing the Files SFILE and OFILE

Initializing the file SFILE is done in exactly the same way as with the file IFILE.

```
1000 REM=====INITIALIZE SFILE=====
1010 GOSUB 500
1020 PRINT"SFILE WILL BE DELETED"
1030 INPUT"TYPE Y TO CONFIRM";A$
1040 IF A$<>"Y"THEN RETURN
BBC:  1045 CLOSE#X:X=OPENOUT F$
      1050 LET C$="E":REM EMPTY
      1060 FOR I=1 TO 50
      1070 LET C$=C$+" ":REM ADD 35 BLANKS
      1080 NEXT I
      1090 FOR I=1 TO 50
Apple: 1100 PRINT D$;"WRITE";F$;",R";I
       1101 PRINT C$
MS:    1100 LSET RC$=C$
       1101 PUT#1,I
CBM:   1100 PRINT#3,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(1)
       1101 PRINT#1,C$
BBC:   1100 PTR#X=38*I
       1101 PRINT#X,C$
       1110 NEXT I
Apple: 1120 PRINT D$;"OPEN";"OFILE"
```

Apple:	1121 PRINT D$;"WRITE";"OFILE" 1130 PRINT D$;"CLOSE";"OFILE"
MS:	1120 OPEN "OFILE"FOR OUTPUT AS#2 1130 CLOSE 2
CBM:	1120 OPEN 2,8,3,"OFILE,S,W" 1121 PRINT#2,"*";REM "*" MEANS FILE EMPTY 1130 CLOSE 2
BBC:	1120 Y=OPENOUT "OFILE" 1130 CLOSE#Y 1140 RETURN

8.3 Entering a new part

Entering a new part is done from line 2000 by a subroutine which must be altered in a few places.

First of all there is the problem with Apple which we have already met: simulating the EOF command by using ONERR confuses the RETURN mechanism, so that a GOTO command has to be used instead of RETURN. This has always worked so far, as the subroutine always **returned to the same location** in the main program. In our new program, however, the subroutine 2300 is called up **at various points.** We get round this with the following programming trick: we set a variable RET at 1 or 2,according to whether we are at subroutine 2000 or 4000 (see lines 2015 and 4015). Using the command

```
2340 ON RET GOTO 2030,4030
```

we jump back to the right point, just as if we were working with the RETURN command.

If you have an Apple computer and want to program without using this trick, you can, of course, look at the instructions given in the third chapter and organize the sequential file OFILE differently. Instead of using EOF, you can, for instance, write the example and write the number of records at the beginning of the file. Then you can process the file OFILE with the help of a FOR-loop and use the ordinary GOSUB/RETURN mechanism. This cuts out the error message OUT OF MEMORY, which otherwise occurs from time to time.

The second alteration concerns the subroutines 2400 and 2500. First we'll look at the reading program from 2500 to 2640. The logical variable OVFL (overflow) is positioned on 0 in line 2510. It is given the value TRUE in line 2560 in case a collision has occurred. The part number contained in the variable PN, which can of course vary from 1 to 999, is transformed into a record number between 1 and 50 by the division method in line 2520.

Now the appropriate record is read. If this is empty, or contains the desired article number, we return to the calling program. Otherwise the file OFILE is searched sequentially. If the desired record is not found, then R$ is marked with the letter "E" (line 2620)

The writing program from line 2400 onwards checks the logical variable OVFL. If no overflow has happened when reading, then the appropriate record is written directly to the file SFILE in the position RN. If, on the other hand, OVFL has the value TRUE, then the record is appended to the file OFILE.

A further alteration is necessary for overwriting a record. We use a logical variable ROVW (remember overwrite), which at first contains the value FALSE (line 2016). In line 2065, this is set to TRUE, if the user wants to overwrite a record. Overwriting in the random file SFILE presents no problems. If the record to be overwritten is in the sequential file OFILE, however, it must first be removed from here. We therefore check the value of ROVW in line 2440, and then, if need be, branch to 2700. There the contents of the file OFILE is copied to the auxiliary file TEMP, with the exception of the record to be overwritten. After this, OFILE is deleted, and TEMP is renamed OFILE (lines 2770 to 2790).

A slight alteration also concerns the construction of the record in R$. In line 2075, the part number is converted into a string of three characters, and this is moved to R$ in line 2100. Please note that, instead of 25, we must now have 20 with the MID$ function. The RETURN command in line 2200 is replaced by a GOTO command with Apple. Finally, when checking PN in line 2320, the relevant figure is no longer 50 but 999.

```
                2000 REM=====CREATE NEW ENTRY=====
                2010 GOSUB 500
Apple:          2015 LET RET=1
                2016 LET ROVW=0:REM REMEMBER OVERWRITE
                2020 GOSUB 2300:REM READ RECORD TO R$
                2030 IF LEFT$(R$,1)="E" THEN 2070
                2040 PRINT"RECORD NOT EMPTY"
                2050 INPUT"OVERWRITE(Y/N)";A$
Others:         2060 IF A$<>"Y"THEN RETURN
Apple:          2060 IF A$<>"Y"THEN 80
                2065 LET ROVW=-1:REM ROVW=TRUE
                2070 LET R$="%":REM RECORD NOT EMPTY
MS,CBM:         2072 LET H$=RIGHT$(STR$(PN),LEN(STR$(PN))-1)
                2075 LET PN$=H$+MID$("   ",1,3-LEN(H$))
                2075 LET PN$= STR$(PN)+MID$("   ",1,3-LEN(STR$(PN)))
                2080 INPUT"DESCRIPTION";DE$
                2090 IF LEN(DE$)>20 THEN PRINT BP$:GOTO 2080
                2100 LET R$=R$+PN$+" "+DE$+MID$(BD$,1,20-LEN(DE$))
                2110 INPUT"UNIT PRICE";U$
                2120 IF LEN(U$)>6 THEN PRINT BP$:GOTO 2110
                2130 LET R$=R$+U$+MID$(BU$,1,6-LEN(U$))
                2140 INPUT"QUANTITY";Q$
                2150 IF LEN(Q$)>4 THEN PRINT BP$:GOTO 2140
                2160 LET R$=R$+Q$+MID$(BQ$,1,4-LEN(Q$))
                2170 INPUT"CORRECT(Y/N)";A$
                2180 IF A$<>"Y"THEN 2070
                2190 GOSUB 2400:REM WRITE RECORD
```

```
Apple:    2200 GOTO 80
Others:   2200 RETURN

          2300 REM======INPUT PART# AND READ RECORD==
          2310 INPUT"PART NUMBER";PN
          2320 IF PN<1 OR PN>999 THEN PRINT BP$:GOTO 2310
          2330 GOSUB 2500:REM READ RECORD
Apple:    2340 ON RET GOTO 2030,4030
Others:   2340 RETURN

          2400 REM WRITE RECORD TO SFILE OR OFILE
          2410 IF OVFL THEN 2440
Apple:    2420 PRINT D$;"WRITE";F$;",R";RN
          2421 PRINT R$
          2422 PRINT D$
MS:       2420 LSET RC$=R$
          2421 PUT#1,RN
CBM:      2420 PRINT#3,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
          2421 PRINT#1,R$
BBC:      2420 PTR#X=38*RN
          2421 PRINT#X,R$
          2430 RETURN
          2440 IF ROVW THEN 2700
Apple:    2450 PRINT D$;"APPEND";"OFILE"
          2451 PRINT D$;"WRITE";"OFILE"
          2460 PRINT R$
          2470 PRINT D$;"CLOSE";"OFILE"
MS:       2450 OPEN "OFILE" FOR APPEND AS#2
          2460 PRINT#2,R$
          2470 CLOSE 2
CBM:      2450 OPEN 2,8,3,"OFILE,S,A"
          2460 PRINT#2, R$
          2470 CLOSE 2
BBC:      2450 Y=OPENUP "OFILE":PTR#Y=EXT#Y
          2460 PRINT#Y,R$
          2470 CLOSE#Y
Apple:    2480 GOTO 80
Others:   2480 RETURN

          2500 REM======READ RECORD TO R$======
          2510 LET OVFL=0:REM OVERFLOW=FALSE
          2520 LET RN=PN-50*(INT(PN/50))+1
Apple:    2530 PRINT D$;"READ";F$;",R";RN
          2531 INPUT R$
          2532 PRINT D$
MS:       2530 GET#1,RN
          2531 LET R$=RC$
CBM:      2530 PRINT#3,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
          2531 INPUT#1,R$
```

```
BBC:        2530 PTR#X=38*RN
            2531 INPUT#X,R$
            2540 IF LEFT$(R$,1)="E" THEN RETURN
            2550 IF VAL(MID$(R$,2,3))=PN THEN RETURN
            2560 LET OVFL=-1:REM OVERFLOW=TRUE
Apple:      2570 PRINT D$;"OPEN";"OFILE"
            2571 PRINT D$;"READ";"OFILE"
MS:         2570 OPEN"OFILE"FOR INPUT AS#2
CBM:        2570 OPEN 2,8,3,"OFILE,S,R"
BBC:        2570 Y=OPENIN "OFILE"
Apple:      2580 ONERR GOTO 2618
MS:         2580 IF EOF(2)THEN 2620
CBM:        2580 IF ST=64 THEN 2620
BBC:        2580 IF EOF#Y THEN 2620
Apple:      2590 INPUT R$
MS,CBM:     2590 INPUT#2,R$
BBC:        2590 INPUT#Y,R$
            2600 IF VAL(MID$(R$,2,3))=PN THEN 2630
            2610 GOTO 2580
Apple:      2618 POKE 216,0
            2619 PRINT D$
            2620 LET R$="E"
Apple:      2630 PRINT D$;"CLOSE";"OFILE"
MS,CBM:     2630 CLOSE 2
BBC:        2630 CLOSE#Y
Apple:      2640 GOTO 2340
Others:     2640 RETURN

Apple:      2700 PRINT D$;"OPEN";"OFILE"
MS:         2700 OPEN"OFILE"FOR INPUT AS#5
CBM:        2700 OPEN 5,8,3,"OFILE,S,R"
BBC:        2700 Y=OPENIN "OFILE"
Apple:      2710 PRINT D$;"OPEN";"TEMP"
MS:         2710 OPEN"TEMP"FOR OUTPUT AS#5
CBM:        2710 OPEN 5,8,4,"TEMP,S,W"
BBC:        2710 Z=OPENOUT"TEMP"
CBM:        2719 LET TS=0
Apple:      2720 PRINT D$;"READ";"OFILE"
            2721 ONERR GOTO 2769
MS:         2720 IF EOF(2)THEN 2770
CBM:        2720 IF TS=64 THEN 2770
BBC:        2720 IF EOF#Y THEN 2770
Apple:      2730 INPUT R1$
MS,CBM:     2730 INPUT#2,R1$
CBM:        2731 LET TS=ST
BBC:        2730 INPUT#Y,R1$
            2740 IF MID$(R1$,2,3)=MID$(R$,2,3)THEN 2720
Apple:      2750 PRINT D$;"WRITE";"TEMP"
            2751 PRINT R1$
```

```
MS,CBM:   2750 PRINT#5,R1$
BBC:      2750 PRINT#Z,R1$
          2760 GOTO 2720
Apple:    2769 POKE 216,0
          2770 PRINT D$;"CLOSE";"OFILE"
          2780 PRINT D$;"DELETE";"OFILE"
          2790 PRINT D$;"RENAME TEMP,OFILE"
          2800 PRINT D$;"CLOSE";"OFILE"
MS:       2770 CLOSE 2:CLOSE 5
          2780 KILL"OFILE"
          2790 NAME"TEMP"AS"OFILE"
          2800 CLOSE 2:CLOSE 5
CBM:      2770 CLOSE 2
          2780 OPEN 4,8,15,"S:OFILE"
          2790 PRINT#4,"R:OFILE=TEMP"
          2800 CLOSE 4:CLOSE 5
          2801 OPEN 1,8,2,F$+",L,"+CHR$(37)
BBC:      2770 CLOSE#Y
          2780 DELETE"OFILE"
          2790 RENAME"TEMP OFILE"
          2800 CLOSE#Y
          2810 GOTO 2450
```

8.4 Unsorted Printing of the Files SFILE and OFILE

First of all the file SFILE is printed out by varying the record number RN from 1 to 50 (line 3040). After this, the records from the file OFILE are printed out. Please note that the records are not sorted according to part number, on account of the scattered storage.

```
          3000 REM=====PRINT SFILE AND OFILE UNSORTED=
          3010 GOSUB 500
          3020 PRINT"CONTENTS OF SFILE"
          3030 PRINT"=================="
          3040 FOR RN=1 TO 50
Apple:    3050 PRINT D$;"READ";F$;",R";RN
          3051 INPUT R$
          3052 PRINT D$
MS:       3050 GET#1,RN
          3051 LET R$=RC$
CBM:      3050 PRINT#3,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
          3051 INPUT#1,R$
BBC:      3050 PTR#X=38*RN
          3051 INPUT#X,R$
          3060 IF LEFT$(R$,1)="E"THEN 3080
          3070 GOSUB 3200:REM DISPLAY RECORD ON SCREEN
          3080 NEXT RN
```

```
                3090 PRINT"CONTENTS OF OFILE"
                3100 PRINT"=================="
Apple:          3110 PRINT D$;"OPEN";"OFILE"
MS:             3110 OPEN"OFILE"FOR INPUT AS#2
CBM:            3110 OPEN 2,8,3,"OFILE,S,R"
BBC:            3110 Y=OPENIN "OFILE"
Apple:          3120 PRINT D$;"READ";"OFILE"
                3121 ONERR GOTO 3159
MS:             3120 IF EOF(2) THEN 3160
CBM:            3120 IF ST=64 THEN 3160
BBC:            3120 IF EOF#Y THEN 3160
Apple:          3130 INPUT R$
                3131 PRINT D$
MS,CBM:         3130 INPUT#2,R$
BBC:            3130 INPUT#Y,R$
                3140 GOSUB 3200
                3150 GOTO 3120
Apple:          3159 POKE 216,0
                3160 PRINT D$;"CLOSE";"OFILE"
MS,CBM:         3160 CLOSE 2
BBC:            3160 CLOSE#Y
                3170 INPUT"CONTINUE(Y/N)";A$
                3180 IF A$<>"Y"THEN 3170
Others:         3190 RETURN
Apple:          3190 GOTO 80
                3200 REM=====DISPLAY RECORD ON SCREEN===
                3210 LET DE$=MID$(R$,2,25)
                3220 LET U$=MID$(R$,27,6)
                3230 LET Q$=MID$(R$,33,4)
                3240 PRINT DE$:PRINT U$:PRINT Q$
                3250 PRINT:PRINT
                3260 RETURN
```

8.5 Access One Part

You can adopt the original program from line 4000 to line 4220. In 4230, we check whether there has been an overflow. If there hasn't, the altered record is simply written back to the random file. If there has, all the records except the one just altered are written from OFILE to TEMP. The new, altered data are to be found in R$, the old, unaltered ones in R1$. With the IF command in 4300, we make sure that either the unaltered records or the altered records are written to TEMP. Then follows the usual renaming of the file TEMP, which becomes OFILE.

```
                4000 REM=====ACCESS ONE PART=====
                4010 GOSUB 500
Apple:          4015 LET RET=2:REM RETURN FOR APPLE
                4020 GOSUB 2300:REM READ RECORD
                4030 IF LEFT$(R$,1)="%"THEN 4080
```

```
4040 PRINT"RECORD EMPTY":PRINT BP$
4050 FOR I=1 TO 1000
4060 NEXT I
Apple:  4070 GOTO 80
Others: 4070 RETURN
4080 GOSUB 3200:REM DISPLAY RECORD ON SCREEN
4090 PRINT"A=ADD TO STOCK"
4100 PRINT"S=SUBTRACT FROM STOCK"
4110 PRINT"R=RETURN TO MENU"
4120 INPUT"YOUR CHOICE";A$
4130 IF A$<>"A"AND A$<>"S"AND A$<>"R"THEN
     4120
Apple:  4140 IF A$="R"THEN 80
Others: 4140 IF A$="R"THEN RETURN
4150 IF A$="S"THEN PRINT"TYPE NEGATIVE NUMBER"
4160 INPUT"QUANTITY";N
4170 LET Q=VAL(Q$)
4180 LET Q=Q+N
4190 IF Q<0 OR Q>9999 THEN PRINT BP$:GOTO 4160
4200 LET Q$=STR$(Q)
MS,CBM: 4201 LET Q$=RIGHT$(Q$,LEN(Q$)-1)
4210 LET S$=LEFT$(R$,32)+Q$+MID$(BQ$,1,4-LEN(Q$))
4220 LET R$=S$
4230 IF OVFL THEN 4260
4240 GOSUB 2400:REM WRITE R$ TO SFILE
Apple:  4250 GOTO 80
Others: 4250 RETURN
Apple:  4260 PRINT D$;"OPEN";"OFILE"
        4270 PRINT D$;"OPEN";"TEMP"
MS:     4260 OPEN"OFILE"FOR INPUT AS#2
        4270 OPEN"TEMP"FOR OUTPUT AS#5
CBM:    4260 CLOSE 1:REM CLOSE RANDOM FILE
        4261 OPEN 2,8,3,"OFILE,S,R"
        4270 OPEN 5,8,4,"TEMP,S,W"
        4271 LET TS=0
BBC:    4260 Y=OPENIN"OFILE"
        4270 Z=OPENOUT"TEMP"
Apple:  4280 PRINT D$;"READ";"OFILE"
        4281 ONERR GOTO 4349
MS:     4280 IF EOF(2) THEN 4350
CBM:    4280 IF TS=64 THEN 4350
BBC:    4280 IF EOF#Y THEN 4350
Apple:  4290 INPUT R1$
        4291 PRINT D$;"WRITE";"TEMP"
MS,CBM: 4290 INPUT#2,R1$
BBC:    4290 INPUT#Y,R1$
        4300 IF MID$(R1$,2,3)<>MID$(R$,2,3)THEN 4330
Apple:  4310 PRINT R$
MS,CBM: 4310 PRINT#5,R$
```

```
BBC:        4310 PRINT#Z,R$
            4320 GOTO 4280
Apple:      4330 PRINT R1$
MS,CBM:     4330 PRINT#5,R1$
BBC:        4330 PRINT#Z,R1$
            4340 GOTO 4280
Apple:      4349 POKE 216,0
            4350 PRINT D$;"CLOSE";"OFILE"
            4360 PRINT D$;"DELETE","OFILE"
            4370 PRINT D$;"RENAME TEMP,OFILE"
            4380 PRINT D$;"CLOSE";"OFILE"
MS:         4350 CLOSE 2
            4360 KILL"OFILE":CLOSE 5
            4370 NAME"TEMP"AS"OFILE"
            4380 CLOSE 2
CBM:        4350 CLOSE 2
            4360 OPEN 4,8,15,"S:OFILE"
            4370 RENAME*TEMP OFILE
            4380 CLOSE 4: CLOSE 5
            4381 OPEN 1,8,2,F$+",L,"+CHR$(37)
BBC:        4350 CLOSE#Y
            4360 DELETE"OFILE"
            4370 CLI$= "RENAME TEMP OFILE":GOSUB
                 10000
            4380 CLOSE#Y
Apple:      4390 GOTO 80
Others:     4390 RETURN

            5000 REM=====CLOSE AND END=====
Apple:      5010 PRINT D$;"CLOSE";F$
MS,CBM:     5010 CLOSE 1
CBM:        5011 CLOSE 3
BBC:        5010 CLOSE#X
            5020 END
```

Again, note that the BBC version of line 4370 calls subroutine 10000 from p.42.

Conclusion

You have now seen the principles of file handling, and have worked with a series of example programs. Now you have come to the end of this book and the beginning of programming.

You can choose the appropriate file structure according to your specific needs. Take the relevant example program as a basis, and construct your own program on it.

If the BASIC version used by your computer differs from the four examples described here, take the one most similar to yours as your example.

And now: good programming!

NOTE

All programs shown in this book can be run on the following microcomputers without any alteration at all:

1. Apple IIe, II+ or IIc
2. IBM-PC (or compatible)
3. VIC 20 with 19.5 KB or Commodore 64 and Floppy 1541
4. BBC

If you are using a different computer, you may have to make a few slight alterations.

Appendix

File handling commands

1. Apple

SAVE	SAVE TEST	Program TEST is stored on disk
	SAVE TEST,D2	Program TEST is stored on disk in drive 2
LOAD	LOAD TEST	Program TEST is loaded from disk in drive 2
CATALOG	CATALOG	Directory is listed
	CATALOG,D2	Directory of disk in drive 2 is listed
DELETE	DELETE TEST	Program TEST is deleted from disk
	DELETE TEST,D2	Program TEST on disk in drive 2 is deleted
OPEN	PRINT CHR$(4);"OPEN";"TEST"	File TEST is opened for input
	PRINT CHR$(4);"READ";"TEST"	
	PRINT CHR$(4);"OPEN" ' "TEST"	File TEST is opened for output
	PRINT CHR$(4);"WRITE";"TEST"	
	PRINT CHR$(4);"APPEND";"TEST"	File TEST is opened for appending new records
	PRINT CHRS(4);"WRITE";"TEST"	
CLOSE	PRINT D$;"CLOSE";"TEST"	Close file TEST
INPUT	INPUT X,Y	Read two values from file
PRINT	PRINT X:PRINT Y	Write two values to file
Random File	PRINT CHR$(4);"OPEN;TEST L20"	Open random file TEST, record length = 20
	PRINT CHR$(4);"READ";"TEST,R";RN	read record, record number contained in RN
	PRINT CHR$(4);"WRITE";TEST,R";RN	write record, record number contained in RN

2. Microsoft

SAVE	SAVE "TEST" SAVE "B:TEST"	Program TEST is stored on disk Program TEST is saved on disk in drive B
LOAD	LOAD "TEST" LOAD "B:TEST"	Program TEST is loaded from disk to workspace Program TEST is loaded from disk in drive B to workspace
FILES	FILES FILES B:	Directory is listed Directory of disk in drive B is listed
KILL	KILL "TEST" KILL "B:TEST"	Program TEST is deleted on disk Program TEST is deleted on disk in drive B
OPEN (first mode)	OPEN"TEST" FOR INPUT AS#1	File TEST is opened for input and assigned logical file number 1
	OPEN"TEST"FOR OUTPUT AS#1	File TEST is opened for output and assigned logical file number 1
	OPEN"TEST"FOR APPEND AS#1	File TEST is opened for appending new records
(second mode)	OPEN"I",#1,"TEST"	File TEST is opened for input and given the logical file number 1
	OPEN"O",#1,"TEST"	File TEST is opened for output and given the logical file number 1
CLOSE	CLOSE#1	Close file previously assigned logical file number 1
INPUT #	INPUT#1,X,Y	Read two values from file
PRINT#	PRINT#1,X	Write to file
EOF	EOF(1)	End of file previously assigned logical file number 1

Random File

FIELD	FIELD#1,10 AS N$	Data buffer definition
LSET	LSET N$="MARY"	Left set
RSET	RSET N$="MARY"	Right set
PUT	PUT#1,RN	Write record, record number contained in RN
GET	GET#1,RN	Read record, record number contained in RN
MKI$	LSET X$=MKI$(K)	Convert integer expression to 2-byte string
MKS$	LSET Y$=MKS$(Y)	Convert single precision expression to 4-byte string
MKD$	LSET Z$=MKD$(Z)	Convert double precision expression to 8-byte string
CVI	LET K=CVI(X$)	Convert 2-byte string to integer value
CVS	LET Y=CVS(Y$)	Convert 4-byte string to single precision value
CVD	LET Z=CVD(Z$)	Convert 8-byte string to double precision value

3. Commodore

SAVE	SAVE "TEST",8 SAVE "@:TEST",8	Program TEST is stored on disk An existing program TEST is overwritten
LOAD	LOAD "TEST",8	Program TEST is loaded from disk to workspace
Directory	LOAD "$",8 LIST	Directory is listed. Existing programs in workspace are deleted
Scratch	OPEN 1,8,15,"S:TEST" CLOSE 1	Program TEST is deleted from disk

OPEN	OPEN 1,8,2,"TEST,S,R"	Sequential file TEST is opened for input and assigned logical number 1 and channel number 2
	OPEN 1,8,2,"TEST,S,W"	Sequential file TEST is opened for output, and assigned numbers as above
	OPEN 1,8,2,"TEST,S,A"	Sequential file TEST is opened for append, and assigned numbers as above
CLOSE	CLOSE#1	Close file TEST
INPUT	INPUT#1,X,Y	Read two values from file TEST
PRINT	PRINT#1,X	Write to file TEST
Random File	OPEN 1,8,2, "TEST,L,"+CHR$(record length + 1)	
	PRINT#1,"P"+CHR$ (channel number) +CHR$ (low) + CHR$ (high) + CHR$ (byte number)	

4. BBC

SAVE	SAVE":1. TEST"	Program TEST is stored on disk in drive 1
LOAD	LOAD":1. TEST"	Program TEST is loaded from disk in drive 1
CAT	*CAT	Directory is listed
DELETE	*DELETE "TEST"	Program TEST is deleted from disk
EOF	EOF#X	Indicates whether end of file has been reached
OPEN	X=OPENOUT "TEST" X=OPENIN "TEST" X=OPENUP "TEST"	File TEST is opened for output File TEST is opened for input File TEST is opened for input or output
CLOSE	CLOSE#X	File TEST is closed

INPUT	INPUT#X,A,B	Read two values from file TEST
PTR	PTR#X=P	Points to record in file
EXT	L=EXT#X	Gives length of written file

Index

APPEND 25 35
Applesoft 14
auxiliary file 51
BASIC versions 12
BBC 15
CATALOG 8
CBM 15
channel number 13
character strings 12
CLOSE 10, 16
collision 123
copying a file 41
customer index 101
CVD 68
CVI 68
CVS 68
data 5
DELETE 8
deleting a record 37
direct access 3, 63
EOF 25
FIELD 65
field 9
file 9
FILES 8
finding a record 35
Floppy disk 1
folding 122
formatting 2
GET 67
hard-sectored 3
hashing 121
header 99
index sequential file 83
index sequential storage 11
Initialisation 2
input 13
INPUT 14 19
insert new record 47
inventory file 121
ISAM 85
KILL 8
linked 11
linked lists 99
LOAD 7
LSET 65
menu 26
Microsoft 14
midsquare 122
MKD$ 68
MKI$ 68
MKS$ 68
ONERR command 32
OPEN 13
output 13
OUTPUT 14
overflow 126
Pascal 5
phone numbers 20, 23
pointer 99
PRINT 17
PUT 67
random access file 63
randomising 121
READ 14
Read mode 14
RENAME 35
RSET 65
SAVE 7
scattered storage 11, 121
school class list 43
sector 2
sector number 3
sequential storage 11
soft-sectored 3
sorted sequential files 43
sorting 95
statistics 54
stock inventory
storage capacity 5
sublist 112
track 2
track number 3
unsorted file 23
WRITE 14
Write mode 14
Write protect notch 1

LIBERATE YOUR MICRO!

--- from the drudgery of cassettes. By investing in one or more disk drives for your computer, you instantly have:

* Faster program loading
* Faster data retrieval
* More reliable storage
* Greatly increased on-line memory

The principle of using disks on your micros are the same, whatever computer you use. So, Michael Chadwick and John Hannah present several principles and explain how to apply them in the following versions of BASIC:

* Commodore - for the VIC 20 and other CBM machines, using the 1541-floppy.
* BBC.
* Microsoft - including PC-DOS on the IBM PC.
* Applesoft

The programs in this book are written in a standard form of BASIC except for the disk commands which are specific to each machine. In each case, the differences are highlighted so that you can choose the correct command for each computer. This unconventional approach broadens the appeal of the book to just about every prospective disk user.

The contents include:

* What to look for in a disk system.
* Reading and writing data with disks.
* Creating sequential files.
* Sorting sequential files.
* Index-sequential files.
* Linked lists.
* Scattered storage.

The book is thorough, and its coverage is vast. But you don't need to be an expert programmer -- easy examples are introduced to illustrate exactly what is happening.

Over 40 example programs are included which demonstrate the principle programming techniques.

GB £ NET +007.50

ISBN 0-905104-83-8

00750

9 780905 104836

Sigma Press
5 Alton Road
Wilmslow
Cheshire.
SK9 5DY

DISK BASIC ON YOUR MICRO

Michael Chadwick &
John Adrian Hannah